Interface

2004
7
Contents



[表紙デザイン：㈱プランニング・ロケッツ]





特集

低消費電力機器からGHzオーダのハイエンド機器まで

MIPSプロセッサ徹底活用研究

Interface

2004
7
Contents

連載「C++によるDSPオブジェクト指向プログラミング」と「TOPPERSで学ぶRTOS技術」は，お休みさせていただきます．

▶ ディスプレイ新製造技術展

SEMI FPD Expo Japan 2004

北村 俊之

「FPDの新世紀を切り拓く」をテーマに，フラット・パネル・ディスプレイ(FPD)の最新技術を展示する「SEMI FPD Expo Japan 2004」が4月7日(水)～9日(金)の3日間，東京ビッグサイトで開催された(写真1)．主催はSEMI．FPD製品は，大型平面テレビ，PDA，携帯電話や自動車，ディジタル家電など情報化社会を支えるヒューマン・インターフェース・デバイスとして高い注目を集めており，この技術進歩は日進月歩である．今年で第4回を迎える同展示会では，世界7か国から138社/団体が出展し，ディスプレイ産業を支える最新の装置や材料，製造技術，製品が一堂に展示されていた．展示会場内は，プロセス/パネル関連，モジュール/検査/設備関連，有機EL/PDP関連の三つのゾーンで構成され，特設コーナとしてアカデミア・コーナが設けられていた．アカデミア・コーナは，千葉大学や東海大学など8大学による出展コーナで，各大学が取り組む最先端技術が展示されていた．

また，出展社セッションを始め，JEITAとSEMI共催によるEDF電子ディスプレイ・フォーラム，電子ディスプレイTutorial，SEMIスタンダード会議などの各種イベントも併せて開催された．同時に，今回で19回目を迎える「EDEX2004 電子ディスプレイ展」(主催：(社)電子情報技術産業協会(JEITA)， 出展社：19社/団体)も開催された．最終的な来場者数は，3日間で24,491人となっていた．



写真1　会場入り口のようす

● SEMI FPD Expo Japan 2004

FAシステムエンジニアリングでは，Pentium M 1.1GHz搭載の小型PC「PC-CUBE」を利用した，DVC/非圧縮カメラからの映像入力，編集，制御，表示までのデモを行っていた(写真2)．同製品は，106.5×68×122.5mmの小型サイズを実現しており，OSにはLinux 2.4またはWindows XP Embeddedの搭載が可能となっている．また，ネットワークは1000Base-Tと100Base-TXを独立して搭載しており，IEEE1394(6ピン)を2ポート，USB2.0，RS-232-C，サウンド・ボードを装備している．


写真2　FAシステムエンジニアリングの小型PC「PC-CUBE」

ノビテックでは，被写体の濃度情報の記録に重点を置いた，計測用高濃度階調・リニア出力カメラ「Densitocam-D/Sシリーズ」(写真3)を出展していた．同製品は，14ビット(16,384)階調の高分解能で詳細な温度変化の記録を可能としている点が大きな特徴であるという．また，入射光に対する出力の直線性は±1%以内の離脱率となっている．出力インターフェースはUSB 2.0に対応しているため，PCとの接続も容易で，PC画面からカメラの制御を行うことも可能となっている．


写真3　ノビテックの計測用高濃度階調・リニア出力カメラ「Densitocam-D/Sシリーズ」

エイコーでは，有機ELの材料およびデバイスなどの研究用に開発された，簡易有機薄膜作成装置「EO-5型」(写真4)の展示を行っていた．同製品は，2室に仕切られた蒸着室で6基の有機セルと電極用Kセルにマスク・ホルダを装備．有機とメタル蒸着を真空を破らずに一貫して行えるよう，まったく新しいコンセプトで開発されている点が大きな特徴とのことであった．


写真4　エイコーの簡易有機薄膜作成装置 EO-5型」

● EDEX2004 電子ディスプレイ展

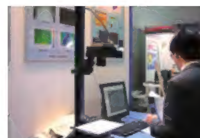
サムスン電子では，今年1月に発表した57インチ液晶ディスプレイ(世界最大)を出展．表示解像度は1920×1080ドットの16：9．独自のPVA(Patterned Vertical Alignment)方式のTFT液晶パネルを採用することで，従来のS-IPS(Super-In-Plane Switching)方式と比較して輝度が20%高まり，コントラスト比の向上(1000：1)，170度以上の広視野角に加えて浅い角度から見ても色調の変化が最小限に抑えられるとのことである．

セイコーエプソンでは，2.1インチ・サイズの高分子有機ELディスプレイを参考出品しており，来場者の高い関心を集めていた．これまでの展示では，携帯電話などをターゲットとしていたが，今回は有機ELの応答速度が高い製品をターゲットにする目的から，モバイル・テレビを選択したとのこと．画面のサイズは2.1インチで208×178ピクセル，26万色の表示を実現しており，消費電力は120mW．このほか，低温ポリシリコンTFTモジュール[LCD(TFT)]技術を応用した「PhotoPC Player P-1000」(写真5)の展示も行っていた．こちらは，約26万色表示の3.8型(640×480ピクセル)ディスプレイを持ち，10Gバイトのハードディスクを搭載したポータブル・ストレージ&ビューワ製品．画像を直接プリンタに出力できるほか，USBホスト機能により，外付けのCD-R/RWドライブにも直接データを保存できるという．

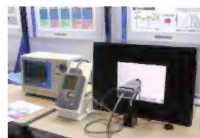

写真5　エプソンのPhotoPC Player P-1000

ワコムでは，携帯電話用のペン入力デバイスの参考出品を行っていた．こちらはタブレットPCでも採用されている同社の電磁誘導式ペン入力デバイスの応用で，有効エリアは46×28.2mm，読み取り高さ5～7mm，最大4個までのペン・スイッチをサポートしているなどの特徴をもつ．ペンの収納スペースなどの課題があり，実用化にはまだ時間がかかりそうだとのことであった．

サイバネットシステムでは，輝度・色度測定システム「ProMetric」(写真6)の展示を行っていた．こちらは，ディスプレイおよびプロジェクタなどの輝度むら，色度むら，欠陥の評価を行うための研究開発，製造ライン用の機器とのこと．測定する分布データは，250,000ポイント以上の解像度で，RGBそれぞれについて高度なキャプチャにより取り込まれ，照明光学系のアライメントに対して解析が可能となっている．


写真6　サイバネットシステムの輝度・色度測定システム「ProMetric」

横河電機では，マルチメディア・ディスプレイ・テスタ「3298F」(写真7)を展示していた．生産ラインでのフリッカ，ホワイト・バランス調整およびガンマ補正データ算出に適しており，電子ディスプレイの輝度，コントラスト，フリッカ，色度をこの1台で測定可能とのことであった．RS-232-Cを標準装備しているため，PCからの制御とデータ通信を行うことも可能．また，低コスト設計により，製品価格を大幅に低減していることも特徴のひとつであるという．


写真7　横河電機のマルチメディア・ディスプレイ・テスタ「3298F」

Show & News Digest

TOPPERSプロジェクト，μITRON4.0仕様フルセット・カーネルを発表

■日時：2004年4月26日（月）
■場所：東実年金会館（東京都中央区）

オープン・ソースのITRON仕様OSを開発しているNPO法人TOPPERSプロジェクトでは，従来のスタンダード・プロファイルのみに対応したTOPPERS/JSPに加え，今回あらたにμITRON4.0フルセット仕様に準拠したTOPPERS/FI4（Fullset of μITRON 4.0）を開発した．FI4はTOPPERSプロジェクトのWebサイト（http://www.toppers.jp/）よりダウンロードできる．

また，同時にOSEK/VDX仕様に準拠したオープン・ソースのTOPPERS/OSEK（仮称）も発表された．OSEKはヨーロッパで高いシェアを誇る自動車制御向けリアルタイムOSである．これは，自動車産業界ではOSEKに対する需要が高いことや，μITRONと機能的に近いこともありTOPPERS/JSPのソース・リストを元に比較的容易に作成できるため，開発したとのことだ．

ほかにはTCP/IPプロトコル・スタックTINETがIPv6に対応したことや，モジュールを動的に追加するためのRLL（Remote Link Loader），TOPPERS C++ APIテンプレート・ライブラリなどが発表された．



TOPPERSプロジェクト 会長 高田 広章氏

第3回UMLロボットコンテスト

■日時：2004年4月13日（火）
■場所：青山TEPIA（東京都港区）

UMLを使ったモデル駆動開発の普及を目的としたロボット・コンテストが開催された．このコンテストはライン・トレース・ロボットを作成し，コースを二周したタイムを競い合うだけでなく，モデル自体も評価の対象となる．モデルは事前審査により6チームに賞が与えられており，モデルの良し悪しとレースの結果が一致するかどうかが注目されていた．結果は，6チームの完走率が25%にとどまり，モデルの良し悪し以前に実装の問題が大きいと感じられた．完走率が100%に近くなったときにはモデルの真価が問われるのではないだろうか．これらのモデル図は会場内に張り出され，参加者どうしでお互いにモデル図を評価する光景も見られた．



コース全体のようす

キャパシタ蓄電システム合弁事業の本格展開で合意

■日時：2004年4月26日（月）
■場所：パレスホテル（東京都千代田区）

オムロン（株），三井物産（株），（株）岡村研究所，（株）パワーシステムは，電気二重層キャパシタ蓄電システム（ECaSS）の本格的な普及を目指して合弁事業を推進していくことで基本合意した．

ECaSSは岡村研究所代表取締役の岡村廸夫氏が開発した蓄電システム．従来の電気二重層キャパシタでは不十分とされていた蓄電量を向上させたもの．今回の合弁事業では，従来ECaSSの事業化を行っていたパワーシステムが5月末をめどに第三者割当増資を行い，オムロンおよび三井物産がこれをすべて引き受け，今後2年間にオムロン，三井物産の両者で20億円規模の投資・融資を実施する予定だ．

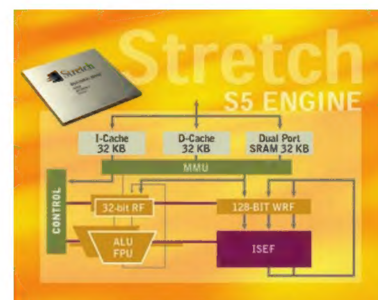


左から三井物産 常務執行役員 阿部 謙氏，オムロン 代表取締役社長 作田 久男氏，岡村研究所 代表取締役 岡村 廸夫氏，パワーシステム 代表取締役社長 山岸 政章氏

Stretch，PLD内蔵プロセッサを発表

■URL：http://www.stretchinc.com/

Stretch社は，PLDを内蔵した組み込み向けプロセッサS5000ファミリを発表した．同プロセッサの開発環境では，プログラム上で負荷となる箇所を見つけだし，それを拡張命令としてコンパイラがハードウェア化し，従来は膨大なコードで実行されていた処理を1命令で処理することが可能になる．これにより，簡単に高速処理を実現することができる．同社では，この新しいプロセッサを，“Software-Configurable Processor”と呼んでいる．



Stretch社のプロセッサS5000ファミリ

ハッカーの常識的見聞録

広畑 由紀夫



今月の常識

グラフィックス・ボードは今年の夏も熱くなる

☆ PCI Express を搭載したマザーボードの発売までしばらく待たされそうな今年の夏ですが，グラフィックス・ボードは昨年以上に熱くなりそうです．今回は nVIDIA 社 GeForce6 に目を向けてみます．

● PCI Express 世代への橋渡し

グラフィックス・ボードは，現行の AGP8x 世代においても，高速なメモリ転送，さらには高度な描画能力を求めるアプリケーションでは限界といわれてきています．そして，バス速度などの大幅な強化がなされる PCI Express までは，今しばらく「待ち」という状態ですが，GPU メーカは着実に PCI Express を搭載したマザーボードを一般ユーザ向けに発売すべく，さらなる強化を図っているようです．

● 強化されたハードウェア

スーパ・スカラ/シェーダ・アーキテクチャ，および 16本のパイプライン処理，さらに GeForce 6800 Ultraでは GDDR3への対応により，GPU の処理能力が大幅に向上していると思われます．具体的な資料はメーカの Webサイト(http://www.nvidia.com/)で確認してもらいたいのですが，GeForce FX 5950 Ultraと仕様上のおもな違いを後で列挙します．

● より強化された対応 API

「Microsoft DirectX 9.0 Shader Model 3.0」への対応および，32ビット浮動小数点テクスチャのサポートが発表されました．さらには，MPEG-1/2/4の CODEC 処理を GPU 内部のプログラマブル・ビデオ・プロセッサ機能を使用して，GeForce 6800のみで行えるとのことです．

単に GPU としての能力のみならず，今後のマルチメディア PC に搭載されることを目指して設計された GPU だといえます．特にこれらの付加価値は Windows XP MediaCenter Edition のようなマルチメディア重視のインターフェースを備えたコンピュータに適しているでしょう．

● GeForce 6800 Ultra と GeForce 6800 のおもな違い

GeForce 6800 Ultraと GeForce 6800のおもな違いは，GeForce 6800では，「パイプラインが 12本」，「サポート・メモリが DDR に制限」，「512M バイト・メモリ版が予定されていない」といったところです．筆者のように，現在，GeForce FX 5950を使っているのであれば，GeForce 6800では物足りないかもしれません．しかし，これから購入を考えているというのであれば，GeForce 6800も考慮に入れられると思います．

● この夏の GPU

nVIDIA 社，GeForce系 GPU の性能強化によって，ATI 社製 GPU の次世代が発表されることは想像にかたくありません．筆者は，この夏，消費電力のアップに気をつけつつ，より性能の高い GPU への買い代えを予定しています．今年の夏も熱くなりそうですが，読者のみなさんもより快適な環境を実現するため，グラフィックス・ボードを買い変えてみませんか．

最後に GeForce FX 5950 Ultraと GeForce 6800 Ultraのおもな性能の違いを参考までに列挙しておきます．

	GeForce 6800 Ultra	GeForce FX 5950 Ultra
Memory Bandwidth (Gバイト/秒)	35.2	30.4
Fill Rate (10億テクセル/秒)	6.4	3.8
Vertices per Second (100万)	600	356
Memory Data Rate (MHz)	1100	950
Pixels per Clock (peak)	16	8



図1　GeForce 6800 GPUの技術的仕様の Web ページ (http://www.nvidia.com/page/pg_20040406661996.html)

ひろはた・ゆきお　OpenLab.

移り気な情報工学

ユーザビリティの視点

山本 強

ユーザビリティ(Usability)に関する話題を聞く機会が多くなった．直訳すれば"使いやすさ"という意味になる．IT分野ではWebページのユーザビリティ評価などでよく使われている．

対語となるのはユーティリティ(Utility)，つまり有用性，機能性である．ユーティリティが機械側の視点で機能を客観的に評価するのに対して，ユーザビリティは人間側の視点で機能を主観，客観の両面から評価するというスタンスの違いがある．

デジカメを例にすると，画素数や撮影モードの数はユーティリティの評価であり，撮影しやすさや収納しやすさはユーザビリティの評価であるといえる．ディジタル家電などのハイテク商品は，発売当初は性能指数で売れるが，開発競争が一段落して性能が横並びになってくるとデザインやユーザビリティが重要になってくるという．だれだって使いやすい物が欲しいに決まっている．開発側も無理に使いにくくしているわけではないのだが，作る側の評価軸と使う側の評価軸は必ずしも一致しているわけではない．エンジニアにユーザビリティを上げろと指示するのは簡単だが，お客様の気持ちがわかればビジネスに失敗はないのである．それが読めないから，みんな苦労しているわけである．とっかかりとして自分が使用者となる分野を取り上げて，ユーザビリティ評価の練習をしてみたい．

ハイテク教室のユーザビリティ評価

大学の教室はいったいだれが設計しているのかと思うことがある．学校の建て替えはめったにあるものではないし，教育を研究する人も研究対象が「教育」なのであって教育を自分でやりたいということではない．教育分野の研究者はIT分野における研究開発部門の立場であり，実際に教室で教えている私などが利用者の立場になる．

今時の教室は液晶プロジェクタ装備で，AVと照明の制御卓が付いているというものが増えている．ユーティリティとしては申し分ないのだが，ユーザビリティ的には問題がある場合が多い．使うまでの準備がたいへんというのはいつものことだが，それは教員も勉強する必要があるということで納得するとしても，それ以前の問題がある．

私が使っている教室の例では，プロジェクタ使用モードと黒板使用モードが完全に分離しており，その移行に2分ほどかかるのだ．切り替えという高機能設備のおかげで，プロジェクタ・モードへの移行で照明が全部切れ，教室が真っ暗になってしまう．これでは学生はノートも取れずに寝てしまうというわけで，部分的に照明のスイッチを入れたいのだが，そのスイッチは制御卓の反対側の入口ドア横に設置されている．

照明も問題で，スクリーンに向かって縦列に3グループに分けてON-OFFするように作られている．教室でプロジェクタを使うとわかるのだが，ON-OFFの単位は横列で前，中，後となっているのが嬉しいのである．もうちょっと考えてくれたら有難い，それがユーザビリティということなのだろう．

私にとっては黒板の真正面にスクリーンが下りてくる形式も使いにくい．放送講座のように完全に作り込まれた教育コンテンツを再生するなら問題はないのだが，そこに教師がいる講義では，プロジェクタ・モードと板書モードの高速切り替えや，スクリーンと黒板の併用は必須なのである．現実はそういう講義がほとんどだと思うのだが，いかがなものだろうか．

ドイツで見かけたユーザビリティの真髄

3月に仕事でドイツに行った際，少し時間ができたので「笛吹き男」で有名なハーメルンに行ってみた．

歴史を感じさせる田舎町なのだが，オフ・シーズンにも関わらずそこそこに観光客がいる．それは良いとして，観光客の歩く流れがまるで笛吹き男に誘導されるように同じ方向を向いて流れているのである．蛇行のぐあいまで似ている．

私と同行した某氏は，ガイドブックも持たずに旧市街に入ってどっちに行けばよいものやらと悩んでいたのだが，ふと足元を見たときに，観光客の流れの意味がわかってしまった．路面にはペンキで点々と「ネズミ・マーク」が書かれていたのである．ネズミ・マークが笛吹き男の故事に関連するランド・マークを結んでいるという簡単な話だった．

ここにユーザビリティの本質が隠れていると思う．何よりもこのシステムは観光の基本である歴史的建築物や街並みを破壊していない．石畳に書かれたネズミ・マークはいずれ消えるものである．言語の壁も越えて意図を伝えている．

ユーザビリティが高いとは，サービスや機能が自然で受け入れやすい形で提供されているということに尽きるのである．ユーザビリティを追求すると物やサービスは単純かつ単機能になるのだろう．

e-Japanが信条の日本では，観光客に情報を伝える機能はITでなければいけないという暗黙の了解があるのではないだろうか．新聞ではGPSや携帯電話がからんだハイテクな観光情報システムの話題を聞くのだが，そういうシステムがそう簡単に普及するようには思えない．

IT業界にいる私としては，ユーザビリティという視点でサービスを見直すと，逆にITが重要ではないという結論が出そうなのがちょっと怖いのである．

やまもと・つよし　北海道大学大学院情報科学研究科
メディアネットワーク専攻

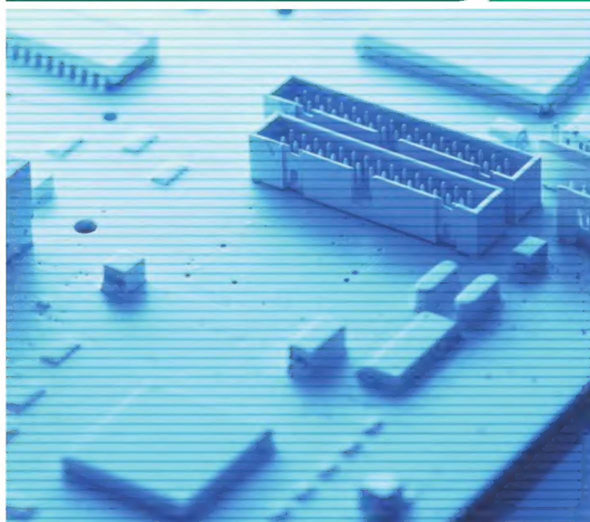
低消費電力機器からGHzオーダのハイエンド機器まで

特集 MIPSプロセッサ徹底活用研究

地上ディジタル放送が開始され，各種STB（セット・トップ・ボックス）/ディジタル・チューナ，そしてHDD/DVDビデオ・レコーダなど，従来の家電とは比較にならないほど高機能な情報家電機器が次々登場している．PCと連携したり，ネットワークに接続できるなど，これら情報家電に要求される処理能力は，これまでの組み込み向けプロセッサでは対応できない性能が要求される．このように，次元の違う処理性能を要求されたとき，どのようなプロセッサを採用すべきだろうか．

MIPSプロセッサは，そのような用途に最適なプロセッサである．200MHzや400MHzといったクロック周波数では，ほかのアーキテクチャのプロセッサよりも消費電力が低い．また下は数十MHzから，上は500MHzや1GHzという周波数まで，多様なMIPSアーキテクチャのプロセッサが供給されている．

この特集では，MIPSの歴史や情報家電機器への採用状況などの現状から，基本的なMIPSアーキテクチャの解説，そして各社のMIPSアーキテクチャ・プロセッサの特徴について解説する．



第1部
プロローグ

なぜMIPSなのか

愛宕 邦夫



MIPSとの出会い

もともと筆者とMIPS系チップとの付き合いはたいへん古く，1994年ごろにIDTのRC30xx（なんと今でもまだ販売している！）でVMEボードを開発したことにさかのぼる．当時は68KなどのCISCからRISCへの流れが組み込み分野にも影響しはじめたころで，コアはまだ32ビットのR3000しかなく，開発はかつてのMIPS社のワークステーション（懐かしい言いかただ）を買うしかなかった．MIPSをサポートしたOSはVxWorksくらいしかなく，Linuxはまだなかった．

何よりシンプルなアーキテクチャ，シンプルな命令セットが気に入り，また実際にスピードも速かった．IDTのRC30xxは価格も安くライフ・サイクルも長そうだったので，RISCではこれで行こうと思って採用した．もっとも，当時デバイスとして簡単に入手できて，ボードとして設計できそうなのは，これくらいしかなかったのである．

それから10年して，なぜ今もMIPSなのかというと，決して過去にこだわってのことではない．あのときと同じようにCPUを選ぼうとしたときに，MIPS系が良かったというだけである．もしこれが3年前だったら，ARM7とSHに対して自信をもってMIPSが良いとはいえなかっただろうが，今ではARMもSHもすでに凌駕していると自信をもって推薦できる．

筆者の知るMIPS系デバイスの中で，現時点でおそらく問題なく入手できるMIPS系のデバイスと，まだ入手はできないが，すでに発表されていてまもなく入手できるだろうデバイスを表1にざっと羅列する．下は60MHzクラスから上はGHzオーダまで，同じアーキテクチャのプロセッサが使えるのである．

表1　現在のMIPS系デバイス

2.5GHz以上	
intrincity	1024点FFTで60万以上をたたきだす，MIPS＋ベクトル演算プロセッサ（残念ながら一般販売はしていない）
800MHzから1GHz	
PMC	RM9000系（CISCOのルータで有名）
NECエレクトロニクス	V_R5500コア（ハイエンド・ストレージやSTB用のコア）
東芝	TX99コア（最新のMIPS64 25Kfコア）
400MHzから800MHz	
AMD/Alchemy	Au1000/1100/1500（400MHzで250mWというSoC）
NECエレクトロニクス	V_R5500/V_R7701
PMC	RM6000系/RM7000系
200MHzから400Mhz	
NECエレクトロニクス	V_R4131/4133（以前のカシオペアなどに使用されていた） EMMA/EMMA2（MIPS32 4K/4KEコア STB向けのSoC）
ATI	Xillion（MIPS32 4K/4KEコア ディジタル・ハイビジョン用STBデバイスのフロント・ランナ）
QuickLogic	QuickMIPS（MIPS32 4Kc＋PCI＋FPGA）
東芝	TX49xx/TX79xx（TX79xxはPS2からの派生プロセッサ）
100MHz前後	
NECエレクトロニクス	V_R4181A（PDA用のSoC）
東芝	TX39系
IDT	RC32364
Philips	NexperiaPNX8系（DVDやSTB用のMIPS＋DSPで構成したSoC）
Brecis	MSP2000/4000/5000（MIPS32 4Km＋DSPのルータ専用SoC）
ESS	VCD/DVDドライブ/プレーヤ用チップ（既存製品はフリーのMIPS-Xを使用 MIPS32 4Kcの製品を発表予定）
ViXIS	ビデオ・ストリーム配信用デバイス（既存製品はフリーのMIPS-XおよびMIPS32 4Kmコアを使用）
ADMtek	超低価格SWHub＆ルータ用SoC（MIPS32 4Kcを使用）
BroadCom	BCM112x/1250/91250など（ネットワーク・プロセッサ）
TI	TNETV1060（VoIPゲートウェイはMIPS32 4KEcコア+DSP）
ZORAN	TL9xx（ディジタルTVソリューション製品はMIPS32 4KEcコア＋DSP）

なぜ今，MIPSなのか

さて，筆者を再度MIPSに突き進ませた最大の要因は，MIPS系全体としては将来とも永らえるだろう，という点である．実はそのことでMIPS社の利益は必ずしも大きく増えないし，MIPS系の勢力の浮き沈みも大きくなるけども．

SHは当然日立（ルネサス）だけがデザインも製造も行う（提携しているSTマイクロで設計/製造するという手もあるが）．PowerPCはIBMとモトローラだけがデザインと製造を行う．これらは昔からのCPUメーカの伝統的なやりかたである．

ARMはこれらとはまったく違い，これを使うメーカはARM社のいくつかのコアをそのままライセンス購入し，周辺だけをデザインして製造する（DECのStrongARMとその後継のIntelのXScaleだけは基本コアの上にデザインする権利を有しているが，これは極めて特殊で高価な契約であり，またARMにもその使用権があるというもの）．

さて，MIPSはARMと同じと思われている人も多いようだ

表2　ARM9とMIPS32 4Kcの比較表

プロセッサ名	プロセス・ルール(μm)	チップ面積(mm²)	動作周波数(MHz)	mW/MHz	200MHz時消費電力(mW)
ARM922T	0.18	8.1	200以上	0.8	160
	0.13	3.2	250以上	0.25	50
ARM940T	0.18	4.2	185以上	0.8	160
MIPS32 4Kc	0.18	1.6～2.5	168～295	0.9 ～1.53	180
	0.13	0.8～1.3	225～394	0.29～0.49	58

Webページなどで公表されている数字を基にしている．MIPSはソフト・コアの場合もあるため一概には比較できない

が，確かにARMと同じくハードウェア・コアでも提供されるが，ソフトウェア合成モデルでの提供も多い．しかし，それよりもなにより特徴なのは，アーキテクチャ・ライセンスと言われるものである．PMC/AMD/NECエレクトロニクス/東芝/ソニー(最新コアはMIPSと共同)などの大手メーカが，アーキテクチャ・ライセンシである．

コアをそのまま使うのではなく，アーキテクチャはMIPSであるが，デザインは自社オリジナルで開発するというもので，言い換えればPowrPCにおけるIBMとモトローラの関係(ライセンス料は考えない)と考えればよいだろうか．

MIPSの顧客

MIPSのコア・ライセンスの顧客には大きく3種類がある．

一つ目はARMなどと同じくユーザ企業が自社のシステムLSIのコアとして採用するケースである．二つ目は，台湾のデザイン・ハウスに多いが，自社の特徴あるDSPやIPコアとMIPSコアを組み合わせ，マーケットを決めてSoCとして作るものである．これはARM7時代にはARMの専売特許であったが，近年はMIPSをコアとして採用したものが増えている．最近のMIPSのリベンジともいえる活躍は，このSoCベンダの活躍によるものである．

最後に三つ目は，MIPSのアーキテクチャを使って自分でコアを設計するものである．これはMIPS特有のものであり，腕に自信のあるベンチャがMIPSを選ぶ大きな理由となっている(将来のMIPSを担保するものであるが)．

MIPS標準コア採用の理由

もともとフリーのMIPS-Xを使っていたベンダは，実はけっこういたのだが(サウンドやDVDコントローラで有名なESSや，MPEG配信で特徴のあるViXISなど)，これはタダだったということが大きなところだったとも思われる．

しかし最近，それらの企業がMIPS純正コアを採用し始めているのは，ARMなどに比べて特に動作周波数が200MHzを超えるような性能の高い範囲では，MIPSのほうが有利と判断されてきているからである．

MMUのない60MHzクラスまででは，ARM7はたいへん小さく，かつ消費電力も少ないためSoCにはうってつけであったが，情報家電や通信機器などで200MHz級が普通に要求されるようになると，PowerPCかMIPSかということになり，コアとして一般売りをしていないPowerPCは採用できず，結果としてMIPSになったということだろう．

ちなみに，この種のコアで問題になるのは，希望の性能が出るという前提のうえでの消費電力とコアのサイズである．筆者がインターネット上で調べたARM9とMIPS32 4Kcの比較を表2に示す．消費電力はほとんど差がなく，コア・サイズはMIPSのほうがかなり小さいのである．

性能の面ではMIPS系はかなり前からARMやSHに対して差をつけていたのだが，いまやSoCコアとしても凌駕できるようになったということではないだろうか．

特に通信がらみと情報家電の分野では，以前からMIPSが多かったのだが，最近はさらに増えている．

速度はMIPSのほうが速いのだろうが，消費電力とダイ・サイズでARMを選んだ人も多いのではないかと思う．また，消費電力はほとんど差がないことも読者の認識とは違うかもしれない．200MHzから400MHzといったクラスでは，ARMよりMIPSのほうがコアも小さく低消費電力になっているのである．

ソフトウェア・プラットホームとして

LinuxやWindows CE，各種リアルタイムOSで，ARMにないものはないのはご存じのとおりだが，MIPSも歴史があることもあり，同じように多くのOSがラインナップされている．最近は特にTOPPERSなどITRON系RTOSにおいても，MIPS用がラインナップされるようにもなってきた．

今後，より利点となると思われる点として，

(1) MIPSはすでにCPUコアの64ビット対応が一般的になっている
(2) ARMと違い古くからMMUサポートが標準だったため，仮想記憶対応OSではMMU対応がなされている

などがあることも付け加えておく．

あたご・くにお　メガソリューション(株)

第1部 Chapter 1 MIPS32およびMIPS64アーキテクチャからシンセサイザブル・コアまで

MIPSアーキテクチャの変遷と現状



中上 一史

本章では，MIPSアーキテクチャの誕生から基本アーキテクチャの変遷，組み込み向けに拡張された機能，SoC向けのシンセサイザブル・コアなど，MIPSアーキテクチャを理解するために必要な基礎知識を解説する．また，2種類のライセンス形態についても紹介する．

（編集部）

1 情報家電分野でのMIPS

2000年に入り，世界中の半導体メーカ各社が，システム・オン・チップ(SoC)を標榜し，プロセス技術のさらなる進歩，パッケージ技術の進化・改良，周辺IPの充実，SoCの協調設計・検証環境の構築とさまざまな取り組みを進めています．しかしその後，未曾有の半導体不況が訪れ，世界の半導体メーカ各社は，生き残りをかけて，いろいろな対応策を模索しました．その結果，各社は自身の持ち味を生かした事業戦略を軸に，持てる技術や営業資産の集中をさまざまな形態で実行してきています．

そのような状況下，2003年秋頃より，各種STB/ディジタル・テレビ，HDD/DVDビデオ・レコーダ，ディジタル・スチル・カメラ，ディジタル・ビデオ・カメラなどで，従来の家電とは比較にならないほど，高性能・高機能な製品が市場に投入され，半導体メーカ各社の業績回復の一端を担ってきています．

これらの新たな組み込みアプリケーションでは，当たり前のように，PCとの連携やネットワークへの接続など，これまでの組み込み向けマイクロプロセッサでは対応できない高い機能が要求されています．また，従来，最終製品のヒットを確定させる“Killer Application”も単一の機能ではなく，より複合的で，複雑なものになってきています．特に情報家電の分野では，消費者のニーズが多様化する中，実生活の一部に，違和感なく，それらのディジタル情報家電が取り込まれつつあります．この動きは消費者が自分のライフ・スタイルに合わせて，適宜に最先端のディジタル情報家電を実生活に取り入れることを示唆しています．これが“Killer Experience”です．

MIPSプロセッサは，消費者が実生活で望む“Killer Experience”を実現できるプロセッサ・コアであり，現在，これらの機器においておもに用いられている動作周波数帯の200MHz～400MHzの領域では，ほかのマイクロプロセッサ・アーキテクチャより低消費電力です．さらに，下は数十MHzから，上は500MHzや1GHzという高い動作周波数帯までを，単一のマイクロプロセッサ・アーキテクチャでカバーできる唯一の存在となっています．

2 組み込みマイクロプロセッサ・アーキテクチャの変遷

1975年10月，IBM社ワトソン研究所で，ハードウェア，コンパイラ，OSの開発が開始され，これが有名なIBM801プロジェクトとなり，米国バークレー大学のSPARCアーキテクチャと米国スタンフォード大学のMIPSアーキテクチャに受け継がれました．まさに，このプロジェクトがRISC型のマイクロプロセッサ・アーキテクチャの出発点であり，その起源といってもよいでしょう．

それ以降，複数の半導体メーカが，さまざまな組み込み向けのRISC型マイクロプロセッサを商品化しました．それぞれのRISC型マイクロプロセッサに盛り込まれた機能ブロックや，その設計思想は，この801プロジェクトの成果を盛り込んでおり，かつ従来の組み込み向けCISC型プロセッサの利点を取り込んで，さらなる進化を遂げてきました．

ここで，801プロジェクトの設計思想について簡単にまとめてみます．

1) ロード/ストア・アーキテクチャをもつ多ポートの大きなレジスタ・バンク
2) 高度なコンパイラ技術への依存
3) 遅延ロード，遅延ストア，遅延分岐を含む単純な命令セットをパイプライン化して1サイクルで実行
4) キャッシュ・メモリの導入

これらの特徴を盛り込んだ商用プロセッサを開発すべく，前述の米国バークレー大学およびスタンフォード大学が精力的に研究を進めた結果，次のような結論を導き出しました．

1) コンパイラは複雑な命令をあまり使用しない
2) 複雑な命令は不合理な性能特性を示す
3) 複雑なマシンは設計に時間がかかる
4) 複雑な命令セットは設計上の誤りを含む確率が高い
5) 複雑な命令はマシン全体を遅くする
6) チップ面積を有効に使用するためにほかにすることがある

特に6番目の項目は，その後のマイクロプロセッサ設計およびSoC化に大きな影響を与えました．つまり，半導体プロセス技術の進歩にともなって，複雑な命令セットを1チップに集積することが可能になりましたが，それに反して単純なプロセッサを開発し，より速いトランジスタ，パイプライン，オンチップ・キャッシュにその領域を取っておくほうが有効であるという考えかたが主流になりました．

3 MIPS基本アーキテクチャの変遷

前述の米国スタンフォード大学の研究成果は，MIPS Computer Systems社と，その半導体ライセンシによって，商用のCMOSプロセスを用いて製品化されました．最初の製品はR2000マイクロプロセッサで，32ビットの整数演算ユニット，例外処理機能，キャッシュ制御機能，メモリ管理機能を1チップ化したデバイスでした．それに外付けの汎用SRAM，浮動小数点演算ユニットR2010，ライト・バッファR2020を用いて高い処理能力をもったUNIXマシンが開発されました(当時のクロック周波数は20MHz)．その際に初めて用いられた命令セットをMIPS I命令セットと呼びます．

その後，さらなるアーキテクチャの改良，拡張が順次行われ，1988年にはR3000マイクロプロセッサ(25MHz)へ進化し，同時に命令セットもMIPS II命令へと進化しました．そして1991年には，従来複数チップだった32ビットの整数演算ユニット，例外処理機能，キャッシュ制御機能，メモリ管理機能，浮動小数点演算ユニット，キャッシュ・メモリ，ライト・バッファなどを1チップに集積するとともに，64ビット化とマルチプロセッサ構成のサポート機能追加が行われ，R4000およびR4400(MIPS II/III命令セット)へと進化しました．1992年には，さらに浮動小数点性能を向上させたR5000が市場に投入され，命令セットもMIPS IV，Vへと進化しました．

しかし，これらのマイクロプロセッサ群をおもに用いたアプリケーションは，UNIXベースのEngineering Workstation(以下EWS)での応用が主で，MIPSアーキテクチャを組み込みアプリケーションに応用展開したのは，おもにMIPSアーキテクチャのライセンシ(東芝，NEC，IDT，LSIロジックなどの半導体メーカ)でした．この第一線の有力な半導体ライセンシは，UNIX EWSを主体として発展してきたMIPSアーキテクチャを，組み込みアプリケーションで応用しやすいように改良を行いました．たとえば，例外処理方法の追加，外部割り込み応答性の強化，ビット/バイト操作を行いやすい命令の追加，積和演算命令の追加などを前述のMIPS I，II，III，IVおよびVの命令セットに対して行い，レーザ・ビーム・プリンタ，複合複写機，ディジタル・スチル・カメラ，PDA，ナビゲーション・システム，ゲーム機器など，初期の広範囲な組み込みアプリケーションにおいて，その応用領域を広めてきました．

MIPS Computer Systems社は，1992年にSGI社のハイエンド・マイクロプロセッサ開発部門として吸収されました．SGIはハイエンド・マイクロプロセッサとして，R10000，R12000などの開発を行い，2次キャッシュ・システムをもったシステムで必要となる命令セットのさらなる拡張，マルチプロセッサ環境で必要となる命令セットの拡張，グラフィックス処理などで必要となるSIMD系の命令セットの拡張をMIPSアーキテクチャに対して行い，MIPS III，IV，V命令に反映させました．

このように，MIPSアーキテクチャは，広範囲な組み込みアプリケーションで応用され，進化してきたわけですが，1998年，現在のMIPS Technologiesが，SGI社からスピン・アウトする際に，1983年より面々と拡張・進化を遂げてきたMIPSアーキテクチャを"MIPS32"アーキテクチャおよび"MIPS64"アーキテクチャとして，再度体系化し，まとめ直しました(図1)．

1998年以降，MIPS Technologiesでは，これらのマイクロプロセッサ・アーキテクチャの技術供与(アーキテクチャ・ライセンス供与)および，それらのアーキテクチャを元に，MIPS Technologiesで開発したフル・シンセサイザブルな32ビット/64ビットのマイクロプロセッサ・コア(ソフト・コア)の技術供与(コア・ライセンス供与)を多数の企業に対して行っています．

4 MIPS基本アーキテクチャの拡張

SGI社からスピン・アウトした後，MIPS Technologiesは，MIPS32およびMIPS64アーキテクチャをより広範囲な組み込みアプリケーションで応用されることを想定して，2001年10月米国で開催されたMicroprocessor Formにて，MIPS32 Release2アーキテクチャおよびMIPS64 Release2アーキテクチャを発表しました．

このRelease2アーキテクチャは，昨今の組み込みアプリケーション応用においてさらに重要視されている次の4点に着目して，拡張を行ったものです．

1) 割り込み応答性のさらなる向上
2) 効率的なビット操作系の命令の拡充
3) メモリ管理方法の拡充
4) コプロセッサ接続の柔軟性の向上

以降では，MIPS32 24Kにも実装した基本アーキテクチャ"MIPS32 Release2"で，それぞれ拡張された点について説明します．



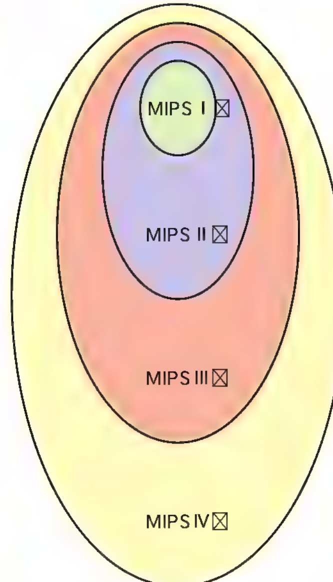


図1 MIPS32およびMIPS64アーキテクチャ
MIPS64アーキテクチャはMIPS32アーキテクチャを完全に包含しており, 完全なソフトウェア互換性を保っている

● 割り込み応答性のさらなる向上

前述したように, 初期のMIPSアーキテクチャはUNIX環境を基本に進化した関係で, 組み込みアプリケーションにおいて要求される外部割り込みに対する処理が, ほかの組み込み型RISCアーキテクチャと比べて良くないといわれていました. そのため, 初期のMIPSアーキテクチャ・ライセンシがさまざまなくふうを凝らしてアーキテクチャを進化させ, それがMIPS32/MIPS64アーキテクチャにまとめられています.

MIPSでは, さらに将来を見据えて, より広範囲な組み込みアプリケーションに応用できるよう, さらなる応答性の向上を図りました.

- リアルタイム・システムにおいては, つねに割り込みの高速応答性が要求されるとともに, その処理時間を特定することが要求されている
- SoC化が進むことにより, SoC内部の割り込み処理は煩雑化・複雑化の一途をたどっている
- ブロードバンド環境が整うにつれてデータの転送能力は格段に向上し, それにともなって, 膨大なパケットを高速に処理するニーズが高まってきている

これらの要求に基づいて, MIPS32 Release2では, 次のことが強化されています.

1) アーキテクチャの規定の中で, 標準的に外付けの割り込み制御方法を規定
2) 優先順位(プライオリティ)付きベクタ割り込みをサポート
3) 汎用シャドウ・レジスタを追加

外部割り込みコントローラは, 発生した割り込み要因のプライオリティを決定し, また, チップ内のプライオリティ・エンコーダ, およびベクタ生成ブロックは, チップ内部の割り込み要因と外部からの割り込み要因を的確に解析して, 特定のベクタを生成します. 一つのコアで, 最大16個の割り込みベクタをサポートできます. 内部割り込み, および外部割り込みの組み合わせ, およびベクタ・アドレスなどは, すべてプログラマブルです. また, 従来のMIPS32/MIPS64アーキテクチャの割り込み制御方法と完全な互換性をもっています(図2).

汎用シャドウ・レジスタ機構は, 割り込みや例外が発生した際に, ハードウェアが自動的にその時点の汎用レジスタの値

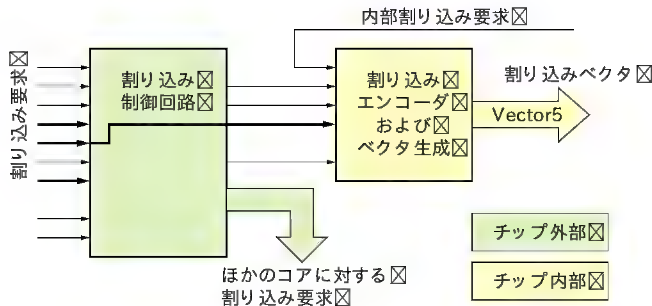


図2 ベクタ割り込み
組み込みシステムで要求される多数の割り込みに対応するために，アーキテクチャ・レベルで外部の割り込み制御方法を規定するとともに，高速な割り込み応答性を実現するために，ベクタ割り込みを規定した．もちろん，従来のMIPS32およびMIPS64の割り込みとも互換性を維持している

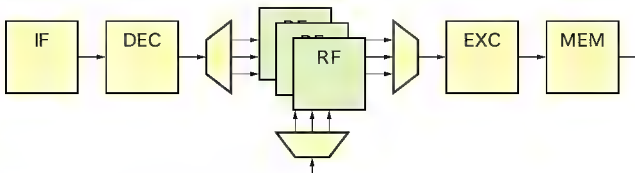


図3 シャドウ・レジスタ
規定したシャドウ・レジスタは，ソフトウェアから見た場合は，従来の単一の汎用レジスタ・セットと同様に見え，レジスタの切り替えをすべてハードウェアで実現している．また，各レジスタに対して割り込みや例外などの割り当てはすべてソフトウェアで明示的にプログラム可能

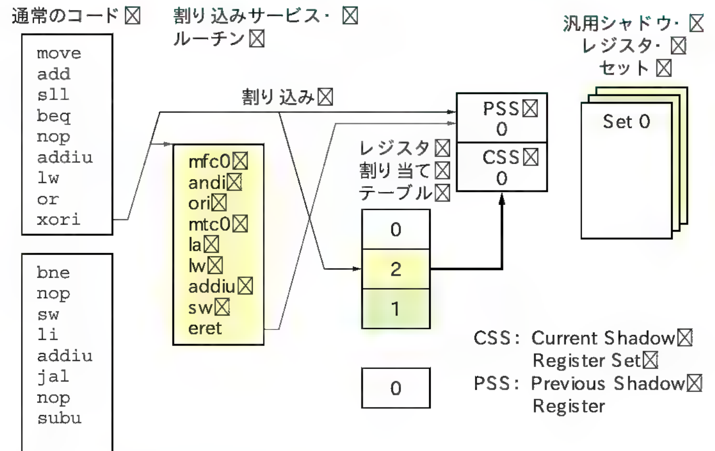


図4 シャドウ・レジスタの動作
割り込みが発生して，対応する割り込みルーチンがサービスされる際に，どのようにシャドウ・レジスタがハードウェア的に切り替わるのかを示している．CSSやPSSは，レジスタ割り当てテーブルの内容に基づいて，レジスタ切り替えを行う

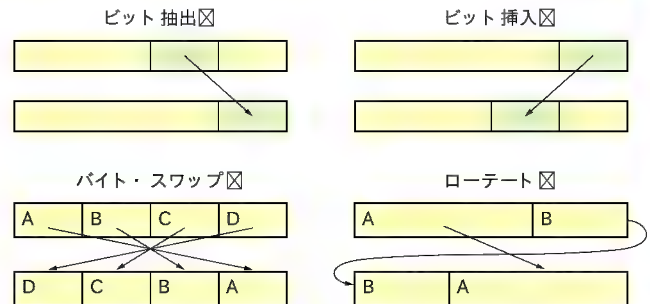


図5 ビット・バイト操作

を，対応するシャドウ・レジスタに保存する機能です．各割り込みや例外をどのシャドウ・レジスタ・セットに対応付けるかは自由に設定できます．アーキテクチャ的には，最大16セットのシャドウ・レジスタを規定できますが，現在，MIPS社で供給しているコアでは，最大4セットのシャドウ・レジスタの搭載ができます(図3，図4)．

● 効率的なビット操作系の命令の拡充

組み込みアプリケーションにおいては，ビット操作，バイト操作など，マイクロプロセッサのデータ処理タイプとは違った操作が要求されます．このあたりの処理に関して，CISC系のマイクロプロセッサがきめ細かい対応を取ってきました．特にブロードバンド環境においては，

- データ転送能力の向上とともに，効率的なプロトコル処理が必要とされている
- より複雑でさまざまなプロトコル処理への対応が必要となってきた

これらの要因に基づいて，MIPS32 Release2アーキテクチャでは，

1) ビット・フィールドに対するビットの挿入/取り出しをサポートする命令の追加
2) バイトのスワップ命令の追加
3) ローテート命令の追加

を行いました．32ビット/64ビット・レジスタに対して，ビットの挿入や抽出を可能とする命令，および同一レジスタ内でデータのエンディアンを切り替え可能な命令，さらに同一のレジスタ内でデータのローテート命令の追加を行い，オリジナルのMIPS32/MIPS64命令で同様のことを行う場合と比べて，大幅な命令数の削減，および処理に必要な一時レジスタの削減を実現しました(図5)．

● メモリ管理方法の拡充

昨今の組み込みアプリケーションのメモリ管理においては，次に示す点の重要性が増してきています．

- 複雑なマルチタスク・ソフトウェアは，独自のメモリ管理方法およびメモリ保護を必要としている
- システム・コストの低価格化には，より効率的なメモリ使用・管理方法を必要としている
- 効率的な仮想メモリ・システムの重要性が増している

これらの要因に基づいて，MIPS32 Release2では，次のことが追加されています．

1) きめ細かいページ・サイズのサポート
2) Contextレジスタに対するプログラマビリティの向上を行い，TLB例外処理時間を削減

最大64Mバイトまたは256Mバイトのページ・サイズをサポートすることにより，大容量のデータ領域を一つもしくは二つのTLBエントリでマップできるようになり，結果として，TLBミスの発生回数を削減し，大容量のデータベースや，テーブルを用いるシステム・パフォーマンスの特定をしやすくしました．

また，1Kバイトおよび2Kバイトのページ・サイズをサポートすることにより，より少ないメモリでも仮想メモリ・システムを構築できるようにしました．もちろん，従来のMIPS32/MIPS64アーキテクチャでサポートしていた最小のページ・サイズである4Kバイトと互換性をもたせることで，過去のソフトウェアとの互換性を保っています．

また，昨今のOSは，それぞれ個別のページ・エントリ・テーブルのフォーマットを持っており，XContextおよびContextレジスタに対するプログラマビリティ性を向上させることにより，TLB例外発生時の処理能力も向上させることが可能になります（図6）．

● コプロセッサ接続柔軟性の向上

アプリケーションの複雑さが高くなるにしたがって，高いレベルでの処理の並列化の必要性が増すとともに，効率的なコプロセッサ処理は，複雑化の一途をたどるシステム構成の単純化，およびコストの削減に絶大な効果を発揮します．そのために，MIPS32 Release2アーキテクチャでは，コプロセッサ接続の柔軟性を図7に示すように向上させ，アプリケーション・パフォーマンスの向上，ダイ・サイズの最適化や，過去の資産の有効活用をしやすくしました．

5 組み込みSoC設計を取り巻く環境の変化

マイクロプロセッサの処理能力は年とともに向上し，また，プロセス技術の進歩と合わせて，有名な“ムーアの法則”にしたがって進化・向上を遂げています．同時にCISC（Complex Instruction Set Computer）とRISC（Reduced Instruction Set Computer）論争を経て，当初は組み込みアプリケーションにRISCプロセッサを応用することにはさまざまなリスクがあると議論されました．

しかし，今日では，RISC型のマイクロプロセッサが非常に広範囲な組み込みアプリケーションに応用され，“Killer Experience”を実現しています．昨今のシステムLSIは，機器のディジタル化，高機能化，さらにはプロセス・テクノロジの進化にともなって，大幅かつ急激に，その複雑さを増しています．

また，1990年代初頭より始まったシステムLSI設計の模索は，ASIC，ASSP，MCM，SIPなど，さまざまな実装，集積化技術の開発を礎として，多数の周辺IPブロックや，32ビット/64ビットのマイクロプロセッサ・コアを用いた“システム・オン・チップ”（SoC）へと展開されています．さらに，半導体のプロセス技術，製造技術の発展にともない，今や，90nmプロセス，300mmウェハの時代を迎えようとしています．

このような時代の流れの中，1998年以降，MIPSは，“高い柔軟性・自由度”，“プログラマビリティ性の向上”，そして“スケーラブルな高い処理能力”を持ったシンセサイザブル・コア，および32ビット/64ビット・マイクロプロセッサ・コア・アーキテクチャの開発を進め，整理統合したものが最新のマイクロ

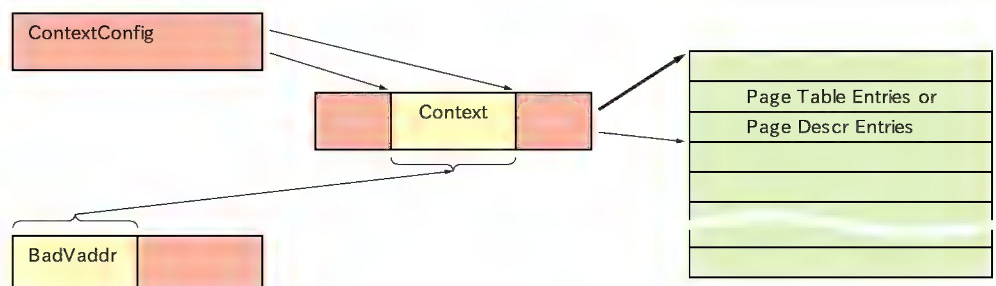


図6
Context および XContext レジスタ
Contextレジスタ，ContextConfigレジスタ，BadVaddrの組み合わせにより，仮想メモリ・システムにおけるPage Table Entryフォーマットと，実際のマイクロプロセッサ内部での仮想アドレス変換にともなうTable Formatを分離できるので，さまざまなOSに対する移植性を高めることができる

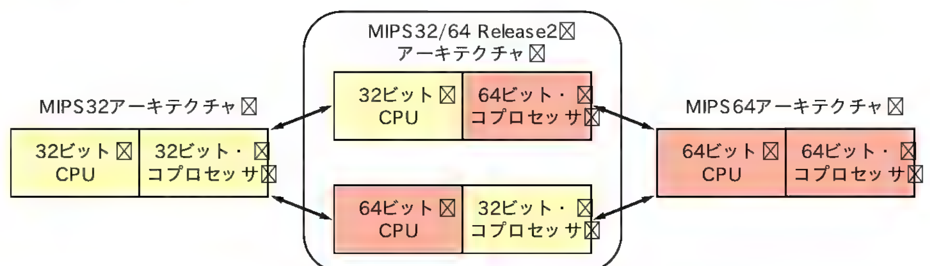


図7　コプロセッサ・サポート拡張
従来32ビットのプロセッサには，32ビットのコプロセッサ，64ビットのプロセッサには，64ビットのコプロセッサ接続しか許されていなかったものを，MIPS32/MIPS64 Release2アーキテクチャでは，32ビットのプロセッサに64ビットのコプロセッサ，64ビットのプロセッサに32ビットのコプロセッサを接続可能にした

プロセッサ・コア"MIPS32 24K"(以下24K)です．MIPSでは，この32ビット・コア・アーキテクチャを，ディジタル家電，ブロードバンド・ネットワーク対応機器市場などへの展開を念頭に，シリーズ化を展開していきます(図8，図9)．

6 MIPSのライセンス形態

● 2種類のライセンス形態

1998年以降，MIPS Technologiesでは，多数の企業に対してアーキテクチャ・ライセンスおよびコア・ライセンスの技術供与を展開しています．現在，アーキテクチャ・ライセンシは14社，コア・ライセンシは97社を数えるに至っています．

この後の説明の理解を容易にするために，まずアーキテクチャ・ライセンシとコア・ライセンシの違いを明確にしておきましょう．

▶ アーキテクチャ・ライセンシ

MIPS32/MIPS64アーキテクチャ準拠の32ビット，および64ビットのマイクロプロセッサ・コアを"独自に"設計開発，製造，販売ができる企業のことをアーキテクチャ・ライセンシと称します．

▶ コア・ライセンシ

ミップス・テクノロジーズが開発したMIPS32/MIPS64アーキテクチャに基づくマイクロプロセッサ・コアを，自社のSoC設計開発において利用することが可能な企業をコア・ライセンシと称します．

● プロセッサ仕様の範囲

では，MIPS32/MIPS64アーキテクチャでは，実際のプロセッサ仕様のどの範囲までを厳格に規定しているのでしょうか？ あまり厳格な規定だと，アーキテクチャ・ライセンシの自由度が損なわれ，また，あまりに大雑把な規定では，さまざまな派生アーキテクチャが誕生し，互換性の面で大きな問題を生じます．

MIPS Technologiesでは，1998年にMIPS32/MIPS64アーキテクチャを再統合・整理をする際に，アプリケーション，およびOSなどのソフトウェアからマイクロプロセッサを見た場合，最低限規定しなければならない要素を，仕様として次のように規定しました．

1) マイクロプロセッサが実行可能な命令群の定義
2) マイクロプロセッサの特権，ユーザ・モードでの動作規定
3) 仮想メモリの規定
4) 例外・割り込み処理の規定
5) コプロセッサの規定
6) システム・リソース制御に関する規定(汎用レジスタ・ファイル，浮動小数点レジスタ・ファイル，浮動小数点演算制御レジスタ群，TLB制御レジスタ，キャッシュ制御レジスタ，各種Statusおよび制御レジスタなど)

上記の規定を遵守することで，コンパイラ，デバッガ，OS，アプリケーション・ソフトウェアなどの互換性を維持しつつ，実際の内部回路の実現手法や独自機能の追加ができる高い自由度をアーキテクチャ・ライセンシに提供することが可能になります．この発想を元に，数百社におよぶサード・パーティ企業のさまざまな設計支援ツール，OS，ミドルウェア，デバッガ，評価ボードをシステム設計に利用できる環境"MIPSエコシステム"が構築されています．

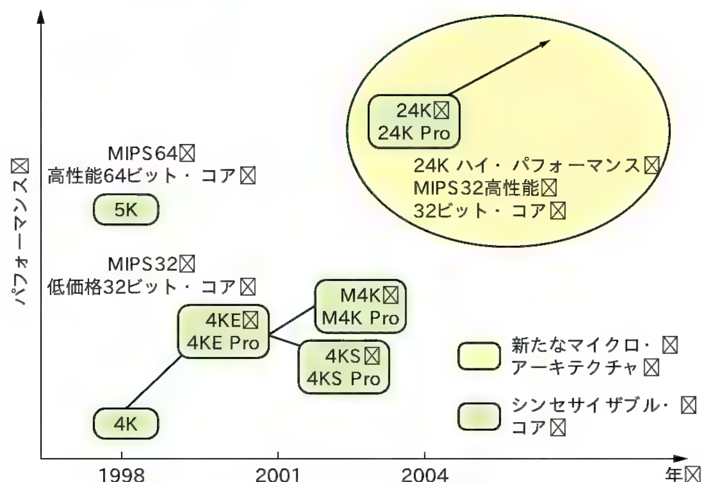


図8　MIPSシンセサイザブル・コア・ロードマップ

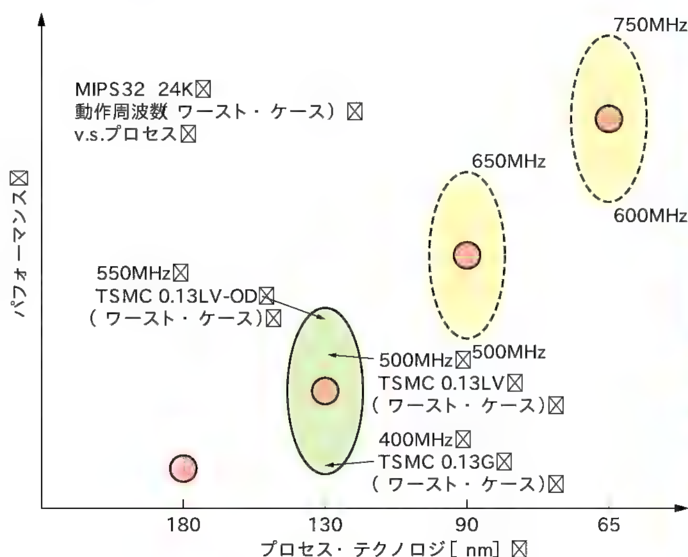


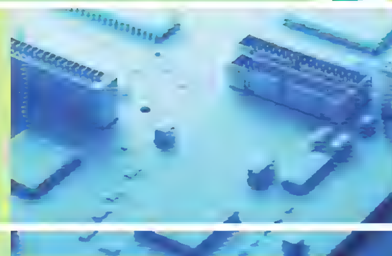
図9　MIPS32 24K動作周波数とプロセス・テクノロジ
LV-ODやLV，GはTSMCの0.13μmプロセスの名称

参考文献
(1)"比較研究RISCアーキテクチャ VLSI RISC Architecture and Organization，基礎から学ぶ，プロセッサ設計とVLSIチップの実例" Stephen B. Furber著，豊橋技術科学大学 今井正治監訳，日経BP社

なかがみ・かずふみ　ミップス・テクノロジーズ

第1部 Chapter 2 命令セット/レジスタ構成からコプロセッサ/例外まで

MIPS32/MIPS64アーキテクチャの詳細



中上 一史

本章ではMIPS32/MIPS64アーキテクチャの命令セットや汎用レジスタの構成，浮動小数点演算ユニット，システム・コプロセッサ0(CP0)について詳しく解説する．特にシステム・コプロセッサ0では，メモリ管理ユニットの動作やTLBについて，各種例外について詳細に述べる．

(編集部)

1 MIPS32/MIPS64アーキテクチャ規定

● MIPS32命令セット

前章の図2で示したように，現在のMIPS32命令セットおよびMIPS64命令セットは，1984年以来培われたMIPS I命令からMIPS IV命令を整理・統合して構成されています．大まかにそれらの命令グループを列記すると以下のようになります．

▶MIPS I命令セット
- ロード/ストア命令
- 算術演算命令
- ジャンプ/分岐命令
- コプロセッサ命令
- スペシャル命令

▶MIPS II命令セット
- トラップ命令
- ロードリンク/条件ストア命令
- 同期命令，特殊分岐命令，ルート演算命令

▶MIPS III命令セット
- 64ビット命令セット拡張(命令長は32ビット)

▶MIPS IV命令セット
- 条件移動命令
- プリフェッチ命令

表1 MIPS32アーキテクチャで規定している命令群

ニモニック	命令	オリジナルMIPS ISAレベル
LB	Load Byte	I
LBU	Load Byte Unsigned	I
SB	Store Byte	I
LH	Load Halfword	I
LHU	Load Halfword Unsigned	I
SH	Store Halfword	I
LW	Load Word	I
SW	Store Word	I
LWL	Load Word Left Format：LWL rt, offset(base)	I
LWR	Load Word Right Format：LWR rt, offset(base)	I
SWL	Store Word Left	I
SWR	Store Word Right	I
LL	Load Linked Word	II
SC	Store Conditional	II
SYNC	Synchronize Memory Operations	II
PREF	Prefetch Memory Data	IV

(a) ロード/ストア/メモリ制御命令

ニモニック	命令	オリジナルMIPS ISAレベル
ADD	Add Word	I
ADDI	Add Immediate Word	I
ADDU	Add Unsigned Word	I
ADDIU	Add Immediate Unsigned Word	I
SUB	Subtract Word	I
SUBU	Subtract Unsigned Word	I
MULT	Multiply Word	I
MULTU	Multiply Unsigned Word	I
DIV	Divide Word	I
DIVU	Divide Unsigned Word	I
SLT	Set on Less Than	I
SLTI	Set on Less Than Immediate	I
SLTU	Set on Less Than Unsigned	I
SLTIU	Set on Less Than Immediate Unsigned	I

(b) 算術演算命令

ニモニック	命令	オリジナルMIPS ISAレベル
AND	Logical AND	I
ANDI	Logical AND Immediate	I
OR	Logical OR	I
ORI	Logical OR Immediate	I
NOR	Logical NOR	I
XOR	Logical XOR	I
XORI	Logical XOR Immediate	I
LUI	Load Upper Immediate Format：LUI rt, immediate	I

(c) 論理演算命令

MIPS32アーキテクチャの基本命令セットは，MIPS II命令セットを基準にしており，それに上記のMIPS IからIV命令よりいくつかの命令を取り込んで構成されています(MIPS II CPU命令，MIPS II FPU命令，CP0命令(特権モード命令)，MIPS IV命令セットより条件ムーブ命令，MIPS IV命令セットよりプリフェッチ命令，単精度浮動小数点フォーマットに対する32本の浮動小数点レジスタに関連する命令，MIPS V命令より浮動小数点制御命令および組み込みアプリケーションで有効ないくつかの命令などを追加)．

表1にMIPS32アーキテクチャで規定している命令群をグループごとにまとめたので参照してください．

● MIPS64命令セット

MIPS64アーキテクチャの基本命令セットは，MIPS V命令セットを基準にしており，それに上記のMIPS IからIV命令よりいくつかの命令を取り込んで構成されています(MIPS V CPU命令，MIPS V FPU命令，CP0命令(特権モード命令)および

表1　MIPS32アーキテクチャで規定している命令群(つづき)

ニモニック	命令	オリジナルMIPS ISAレベル
MOVN	Move Conditional on Not Zero	IV
MOVZ	Move Conditional on Zero	IV
MOVF	Move Conditional on FP False	IV
MOVT	Move Conditional on FP True	IV
MFHI	Move from HI	I
MFLO	Move from LO	I
MTHI	Move to HI	I
MTLO	Move to LO	I

(d) 移動命令

ニモニック	命令	オリジナルMIPS ISAレベル
SLL	Shift Word Left Logical	I
SLLV	Shift Word Left Logical Variable	I
SRL	Shift Word Right Logical	I
SRLV	Shift Word Right Logical Variable	I
SRA	Shift Word Right Arithmetic	I
SRAV	Shift Word Right Arithmetic Variable	I

(e) シフト演算命令

ニモニック	命令	オリジナルMIPS ISAレベル
BEQ	Branch on Equal	I
BNE	Branch on Not Equal	I
BLEZ/BGEZ	Branch on Less/Greater Than or Equal Zero	I
BLTZ/BGTZ	Branch on Less/Greater Than Zero	I
BGEZAL	Branch on Greater Than or Equal 0 & Link	I
BLTZAL	Branch on Less Than or Equal 0 & Link	I
J	Jump	I
JAL	Jump and Link	I
JR	Jump Register	I
JALR	Jump and Link Register	I

(f) 分岐・ジャンプ命令

ニモニック	命令	オリジナルMIPS ISAレベル
BEQL	Branch on Equal Likely	II
BNEL	Branch on Not Equal Likely	II
BLEZL	Branch on Less Than or Equal Zero Likely	II
BGEZL	Branch on Greater Than or Equal 0 Likely	II
BLTZL	Branch on Less Than Zero Likely	II
BGTZL	Branch on Greater Than Zero Likely	II
BGEZALL	Branch on Greater Than or Equal 0 & Link	II
BLTZALL	Branch on Less Than or Equal 0 & Link	II

(g) 特殊分岐命令

ニモニック	命令	オリジナルMIPS ISAレベル
SYSCALL	System Call	I
BREAK	Breakpoint	I
TEQ/TNE	Trap if Equal/Not Equal	II
TEQI/TNEI	Trap if Equal/Not Equal Immediate	II
TLT	Trap if Less Than	II
TLTI	Trap if Less Than Immediate	II
TLTU	Trap if Less Than Unsigned	II
TLTIU	Trap if Less Than Immediate Unsigned	II
TGE	Trap if Greater Than or Equal	II
TGEI	Trap if Greater Than or Equal Immediate	II
TGEU	Trap if Greater Than or Equal Unsigned	II
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	II

(h) トラップ命令

ニモニック	命令	オリジナルMIPS ISAレベル
MADD/MADDU	Multiply and Add Word	N/A
MSUB/MSUBU	Multiply and Subtract Word/Unsigned	N/A
MUL	Multiply Word to Register	N/A

(i) 追加命令

ニモニック	命令
CACHE	Performs cache operations
ERET	Exception Return
MFC0	Move from Coprocessor Zero
MTC0	Move to Coprocessor Zero
TLBP	TLB命令
TLBR	TLB命令
TLBWI	TLB命令
TLBWr	TLB命令
WAIT	Enter Standby Mode
DERET	EJTAG命令
SDBBP	EJTAG命令

(j) CP0命令，EJTAG命令，COP2命令

表2　MIPS64アーキテクチャで規定している命令群

ニモニック	命　令	オリジナルMIPS ISAレベル	備　考
LB/LBU	Load Byte/Load Byte Unsigned	I	
SB	Store Byte	I	
LH/LHU	Load Halfword/ Load Halfword Unsigned	I	
SH	Store Halfword	I	
LW	Load Word	I	
SW	Store Word	I	
LD	Load DWord	III	64ビット
SD	Store DWord	III	64ビット
LWL/LWR	Load Word Left/Right	I	
LDL/LDR	Load DWord Left/Right	III	64ビット
SWL/SWR	Store Word Left/Right	I	
SDL/SDR	Store DWord Left/Right	III	64ビット
LL	Load Linked Word	II	
LLD	Load Linked DWord	III	64ビット
SC	Store Conditional	II	
SCD	Store Conditional DWord	III	64ビット
SYNC	Synchronize Memory Operations	II	
PREF	Prefetch Memory Data	IV	

（a）ロード/ストア/メモリ制御命令

ニモニック	命　令	オリジナルMIPS ISAレベル
AND	Logical AND	I
ANDI	Logical AND Immediate	I
OR	Logical OR	I
ORI	Logical OR Immediate	I
NOR	Logical NOR	I
XOR	Logical XOR	I
XORI	Logical XOR Immediate	I

（d）論理演算命令

ニモニック	命　令	オリジナルMIPS ISAレベル
MOVN	Move Conditional on Not Zero	IV
MOVZ	Move Conditional on Zero	IV
MFHI	Move from HI	I
MFLO	Move from LO	I
MTHI	Move to HI	I

（e）移動命令

ニモニック	命　令	オリジナルMIPS ISAレベル
ADD	Add Word	I
ADDI	Add Immediate Word	I
ADDU	Add Unsigned Word	I
ADDIU	Add Immediate Unsigned Word	I
SUB	Subtract Word	I
SUBU	Subtract Unsigned Word	I
MULT	Multiply Word	I
MULTU	Multiply Unsigned Word	I
DIV	Divide Word	I
DIVU	Divide Unsigned Word	I
SLT	Set on Less Than	I
SLTI	Set on Less Than Immediate	I
SLTU	Set on Less Than Unsigned	I
SLTIU	Set on Less Than Immediate Unsigned	I

（b）算術演算命令（32ビット・データ）

ニモニック	命　令	オリジナルMIPS ISAレベル	備　考
SLL	Shift Word Left Logical	I	
SLLV	Shift Word Left Logical Variable	I	
SRL	Shift Word Right Logical	I	
SRLV	Shift Word Right Logical Variable	I	
SRA	Shift Word Right Arithmetic	I	
SRAV	Shift Word Right Arithmetic Variable	I	
DSLL	Shift DWord Left Logical	III	64ビット
DSLLV	Shift DWord Left Logical Variable	III	64ビット
DSLL32	Shift DWord Left Logical Plus 32	III	64ビット
DSRL	Shift DWord Right Logical	III	64ビット
DSRLV	Shift DWord Right Logical Variable	III	64ビット
DSRA	Shift DWord Right Arithmetic	III	64ビット
DSRAV	Shift DWord Right Arithmetic Variable	III	64ビット
DSRA32	Shift DWord Right Arithmetic Plus 32	III	64ビット
DSRL32	Shift DWord Right Logical Plus 32	III	64ビット

（f）シフト命令

ニモニック	命　令	オリジナルMIPS ISAレベル	備　考
DADD	Add DWord	III	64ビット
DADDI	Add Immediate DWord	III	64ビット
DADDU	Add Unsigned DWord	III	64ビット
DADDIU	Add Immediate Unsigned DWord	III	64ビット
DSUB	Subtract DWord	III	64ビット
DSUBU	Subtract Unsigned DWord	III	64ビット
DMULT	Multiply DWord	III	64ビット
DMULTU	Multiply Unsigned DWord	III	64ビット
DDIV	Divide DWord	III	64ビット
DDIVU	Divide Unsigned DWord	III	64ビット

（c）算術演算命令（64ビット・データ）

ニモニック	命　令	オリジナルMIPS ISAレベル
BEQ	Branch on Equal	I
BNE	Branch on Not Equal	I
BLEZ/ BGEZ	Branch on Less/ Greater Than or Equal Zero	I
BLTZ/ BGTZ	Branch on Less/ Greater Than Zero	I
BGEZAL	Branch on Greater Than or Equal 0 & Link	I
BLTZAL	Branch on Less Than or Equal 0 & Link	I
J	Jump	I
JAL	Jump and Link	I
JR	Jump Register	I
JALR	Jump and Link Register	I

（g）分岐・ジャンプ命令

表2 MIPS64アーキテクチャで規定している命令群(つづき)

ニモニック	命令	オリジナル MIPS ISA レベル
BEQL	Branch on Equal Likely	II
BNEL	Branch on Not Equal Likely	II
BLEZL	Branch on Less Than or Equal 0 Likely	II
BGEZL	Branch on Greater Than or Equal 0 Likely	II
BLTZL	Branch on Less Than Zero Likely	II
BGTZL	Branch on Greater Than Zero Likely	II
BGEZALL	Branch on Greater Than or Equal 0 & Link	II
BLTZALL	Branch on Less Than or Equal 0 & Link	II

(h)特殊分岐命令

ニモニック	命令	オリジナル MIPS ISA レベル
SYSCALL	System Call	I
BREAK	Breakpoint	I
TEQ/TNE	Trap if Equal/Not Equal	II
TEQI/TNEI	Trap if Equal/Not Equal Immediate	II
TLT	Trap if Less Than	II
TLTI	Trap if Less Than Immediate	II
TLTU	Trap if Less Than Unsigned	II
TLTIU	Trap if Less Than Immediate Unsigned	II
TGE	Trap if Greater Than or Equal	II
TGEI	Trap if Greater Than or Equal Immediate	II
TGEU	Trap if Greater Than or Equal Unsigned	II
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	II

(i)トラップ命令

ニモニック	命令	オリジナル MIPS ISA レベル	備考
MADD/MADDU	Multiply and Add Word	MIPS32	
MSUB/MSUBU	Multiply and Subtract Word/Unsigned	MIPS32	
MUL	Multiply Word to Register	MIPS32	
CLZ/CLO	Count Leading Zeros/Ones	MIPS32	
DCLZ/DCLO	Count Leading Zeros/Ones in a DWord	MIPS32	64ビット
SSNOP	Superscalar Inhibit NOP	MIPS32	

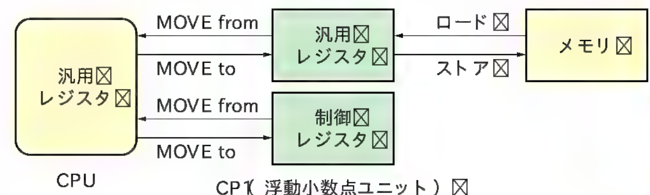
(j)追加命令

ニモニック	命令	備考
CACHE	Performs cache operations	
ERET	Exception Return	
MFC0	Move from Coprocessor Zero	
MTC0	Move to Coprocessor Zero	
DMFC0	Double Move from Coprocessor Zero	64ビット
DMTC0	Double Move to Coprocessor Zero	64ビット
TLBP	TLB命令	
TLBR	TLB命令	
TLBWI	TLB命令	
TLBWr	TLB命令	
WAIT	Enter Standby Mode	
DERET	EJTAG命令	
SDBBP	EJTAG命令	
COP1 xxx	拡張命令(FPU命令)	

(k)CP0命令,EJTAG命令,COP1命令

表3 MIPS32/MIPS64 Release2アーキテクチャで規定している命令群

ニモニック	命令
SEB	Sign-Extend Byte
SEH	Sign-Extend Halfword
EXT	Extract Bit Field
INS	Insert Bit Field
WSBH	Word Swap Byte Within Halfwords
ROTR	Rotate Word Right
ROTRV	Rotate Word Right Variable
JALR.HB	Jump and Link Register with Hazard Barrier
JR.HB	Jump Register with Hazard Barrier
EHB	Execution Hazard Barrier
SYNCI	Sync Caches to Make Instruction Writes Effective
RDHWR	Read Hardware Register
DI	Disable Interrupts
EI	Enable Interrupts
RDPGPR	Read GPR from Previous Shadow Set
WRPGPR	Write GPR to Previous Shadow Set



・データ転送命令
・算術演算命令
・データ変換命令
・フォーマット・データ移動命令
・条件分岐命令
・その他の命令

図1 浮動小数点演算命令グループ

組み込みアプリケーションで有効ないくつかの命令を追加].

表2にMIPS64アーキテクチャで規定している命令群をグループごとにまとめたので参照してください.

● MIPS32/MIPS64 Release2 拡張命令セット

表3にMIPS32/MIPS64 Release2アーキテクチャにおいて,追加拡張された命令セット群を示します.Release2アーキテクチャでは,以下の4点に主眼をおいて命令セットの拡充を行いました.

1) 割り込み応答性のさらなる向上
2) 効率的なビット操作系の命令の拡充
3) メモリ管理方法の拡充
4) コプロセッサ接続の柔軟性の向上

● 浮動小数点命令セット

MIPS32/MIPS64アーキテクチャでは,浮動小数点演算用の命令をCP1命令として規定しています.図1に浮動小数点演算命令グループを示します.表4(a)に示した浮動小数点データ移動・転送命令は,

表4　浮動小数点演算命令群

ニモニック	命　令	オリジナル MIPS ISA レベル
MTC1	Move Word To FPR	MIPS32
MFC1	Move Word From FPR	MIPS32
DMTC1	Move Double Word to FPR	MIPS64
DMFC1	Move Double Word from FPR	MIPS64
CTC1	Copy Word To FP Control Register	MIPS32
CFC1	Copy Word From FP Control Register	MISP32
LWC1/ SWC1	Load/Store FP Word From/ to memory	MIPS32
LWXC1/ SWXC1	Load/Store FP Word Indexed From/ to memory	MIPS64
LDC1/ SDC1	Load/Store FP DWord From/ to memory	MIPS32
LDXC1/ SDXC1	Load/Store FP Word Indexed From/ to memory	MIPS64
LUXC1/	Load/Store FP Word Indexed	MIPS64

（a）FPUデータ転送命令

ニモニック	命　令	オリジナル MIPS ISA レベル
MOV.fmt	Move	MIPS32
MOV.fmt（PS）	Move（paired single）	MIPS64
MOVF.fmt	Move conditional on FP false	MIPS32
MOV.fmt （PS）	Move conditional on FP false （paired single）	MIPS64
MOVT.fmt	Move conditional on FP true	MIPS32
MOVT.fmt （PS）	Move conditional on FP true （paired single）	MISP64
MOVN.fmt	Move conditional on GPR non 0	MIPS32
MOVN.fmt （PS）	Move conditional on GPR non 0 （paired single）	MIPS64
MOVZ.fmt	Move conditional on GPR equal 0	MIPS32
MOVZ.fmt （PS）	Move conditional on GPR equal 0 （paired single）	MIPS64

（d）FPUデータ移動命令

ニモニック	命　令	オリジナル MIPS ISA レベル
ABS.fmt（PS）	Absolute value（paired single）	MIPS32/MIPS64
ADD.fmt（PS）	Add（paired single）	MIPS32/MIPS64
C.cond.fmt /（PS）	Compare（paired single）	MIPS32/ MIPS64
DIV.fmt	Divide	MIPS32
MUL.fmt（PS）	Multiply（paired single）	MIPS32/MIPS64
NEG.fmt（PS）	Negate（paired single）	MIPS32/MIPS64
SQRT.fmt	Square root	MIPS32
SUB.fmt（PS）	Subtract（paired single）	MIPS32/MIPS64
RECIP.fmt	Reciprocal approximation	MIPS64
RSQRT.fmt	Reciprocal square root approximation	MIPS64
MADD.fmt（PS）	Multiply add（paired single）	MIPS64
MSUB.fmt（PS）	Multiply subtract /（paired single）	MIPS64
NMADD.fmt /（PS）	Negative multiply add /（paired single）	MIPS64
NMSUB.fmt /（PS）	Negative multiply subtract /（paired single）	MIPS64

（b）FPU算術演算命令

ニモニック	命　令	オリジナル MIPS ISA レベル	備　考
BC1F.fmt	Branch on FP false	MIPS32	
BC1T.fmt	Branch on FP true	MIPS32	
BC1FL.fmt	Branch on FP false Likely	MIPS32	廃止

（e）FPU条件分岐命令

ニモニック	命　令	オリジナル MIPS ISA レベル
MOVN	Move conditional on FP false	MIPS32
MOVZ	Move conditional on FP true	MIPS32
ALNV.PS	Align a paired single	MIPS64
PLL.PS	Pair lower lower	MIPS64
PLU.PS	Pair lower upper	MIPS64
PUL.PS	Pair upper lower	MIPS64

（f）FPUその他命令

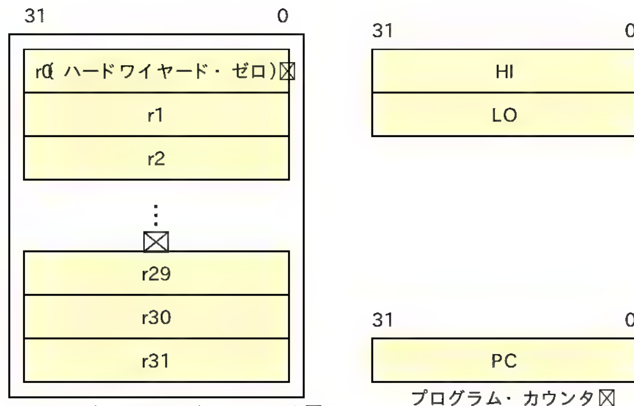
ニモニック	命　令	オリジナル MIPS ISA レベル
CVT.D.fmt	Convert to double FP	MIPS32
CVT.L.fmt	Convert to long fixed point	MIPS64
CVT.PS.fmt	Convert pair to paired single	MIPS64
CVT.S.fmt	Convert to single FP	MIPS32
CVT.S.fmt （PL，PU）	Convert to single FP （Paired Lower, Paired Upper）	MIPS64
CVT.W.fmt	Convert to word fixed point	MISP32
CEIL.L.fmt	Ceiling to long fixed point	MIPS64
CEIL.W.fmt	Ceiling to word fixed point	MIPS32
FLOOR.L.fmt	Floor to long fixed point	MIPS64
FLOOR.W.fmt	Floor to word fixed point	MIPS32
ROUND.L.fmt	Round to long fixed point	MIPS64
ROUND.W.fmt	Round to word fixed point	MIPS32
TRUNC.L.fmt	Truncate to long fixed point	MIPS64

（c）FPUデータ変換命令

- ロード・ストア命令は，データをメモリから浮動小数点レジスタ間で受け渡す
- 移動命令は，データを浮動小数点レジスタから CPU 汎用レジスタ間で受け渡す
- 制御命令は，データを浮動小数点制御レジスタと CPU 汎用レジスタ間で受け渡す
- 上記すべてのデータ処理は，データのフォーマット処理などを一切行わない状態で実行される

表4（b）に示した浮動小数点算術演算命令は，すべてフォーマットに従ったデータ演算を行います．

表4（c）に示した浮動小数点データ変換命令は，浮動小数点データと固定小数点データなどのデータ・タイプの変換を行います．データ変換の際のデータの丸め込みは，FCSR（RM）ビットの設定に従って行うか，命令で明示的に示された指示に従っ



r0 ：ハード・ワイヤード"0"レジスタで，何をストアしても，つねに"0".
r31：命令セットにて明示的に指定されない限り，JAL, BLTZAL, BLTZALL, BGEZAL および BGEZALL 命令のデスティネーション・レジスタとして使用される．それ以外は汎用レジスタとして使用可能.
HI ：乗算，除算結果の上位を保持するレジスタ
LO：乗算，除算結果の下位を保持するレジスタ

図2　CPUレジスタ構成

MIPSアーキテクチャにおける汎用レジスタの規定は実にシンプルである．これは前提としてハードウェアで実装された汎用レジスタを，コンパイラが一義的にレジスタに意味づけをして効率良く使用することを想定しているためである

表5　Cコンパイラでのレジスタ使用方法

すべてのMIPS系のマイクロプロセッサでは，上記のように32本の汎用レジスタそれぞれに意味付けを持たせて，効率的にコンパイラで使用している．

レジスタ名	Software Name	レジスタ定義
$0	zero	ゼロ・レジスタ(つねに"0"を返す)
$1	at	アセンブラ用の一時レジスタ(アセンブラ用に予約済み)
$2..$3	v0,v1	ファンクションからのリターン変数(整数)
$4..$7	a0～a3	ファンクションへの引き数(四つの整数変数)
$8..$15	t0-t7	一時レジスタ
$16..$23	s0-s7	保持レジスタ
$24..$25	t8,t9	一時レジスタ
$26..$27	$kt0～$kt1	例外処理用レジスタ(OS用に予約済み)
$28 or $gp	gp	グローバル・データ・ポインタ
$29 or $sp	sp	スタック・ポインタ
$30 or $fp	s8/fp	フレーム・ポインタ(保持用)
$31	ra	リターン・アドレス・レジスタ

て行われます．

表4(d)に示した浮動小数点フォーマット値移動命令は，浮動小数点レジスタ間での，フォーマット化されたデータの受け渡しを行う際に用いられます．データの移動に際し，無条件移動，もしくは条件移動を行うことが可能です．

表4(e)に示した浮動小数点条件分岐命令は，FCSR内にある8条件との比較結果に応じて分岐を実行します．

表4(f)は，FPUの条件によりCPUの汎用レジスタへデータを転送したり，データ対のアライン，もしくはマージを行う命令です．

2　汎用レジスタの構成

MIPS32/MIPS64アーキテクチャでは，図2に示す32ビット幅の32本の汎用レジスタ・セットを最低1セット規定する必要があります．さらに最新のMIPS32/MIPS64 Release2アーキテクチャでは，最大16セットの汎用レジスタ・セットをもつことが可能です．

これらの汎用レジスタ群は，表5に示すように，現在商用で使用されているCコンパイラおよびGNU Cコンパイラにおいて，各レジスタの使用意味合いが統一されており，効率的なコード生成の一助になっています．

3　CP1 浮動小数点レジスタ

MIPS32/MIPS64アーキテクチャでは，32本の64ビット浮

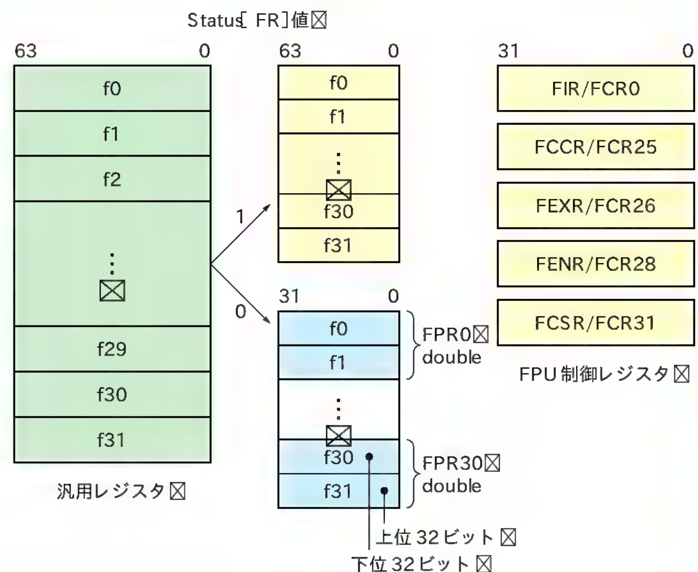
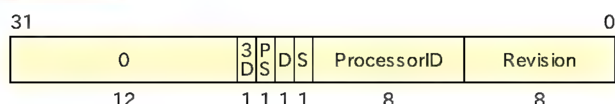


図3　CP1レジスタ群

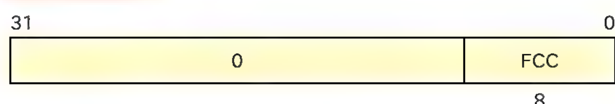
CP0のStatusレジスタのFRビットによって，64ビット幅の浮動小数点レジスタの扱い方を変更することが可能．
FR=1の場合は，32本の64ビット浮動小数点レジスタ．
FR=0の場合は，32本の32ビット浮動小数点レジスタで，偶数レジスタおよび奇数レジスタの二つで，64ビット浮動小数点データを保持するので，16個の64ビット浮動小数点データを扱うこととなる

動小数点レジスタFPRが規定されています．CP0に属するStatusレジスタのFRビットの設定に応じて，32本の64ビット浮動小数点レジスタ(FR=1の場合)，もしくは32本の32ビット浮動小数点レジスタ(FR=0の場合)として使用されます．32ビット幅の浮動小数点レジスタとして使用する場合は，32ビットの偶数レジスタと奇数レジスタ二つを用いて64ビット浮動小数点データを扱うことになるので，16個の64ビット浮動小数



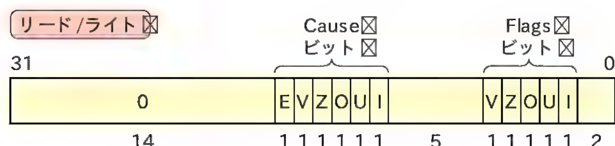
3D : Indicates if the FPU supports MIPS-3D
PS : Indicates if PS (paired single) data types are supported
D : Indicates if D (double) data types are supported
S : Indicates if S (single) data types are supported
ProcessorID : FP processor identification
Revision : Specifies the revision number of the FPU
0 : Must be written 0, returns 0 on read

（a）FIRレジスタ



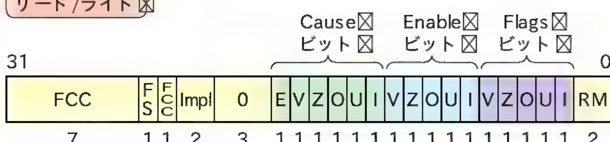
FCC : Fp Condition Codes (used in conditional branches and moves)
0 : Must be written 0, returns 0 on read

（c）FCCRレジスタ



Cause bits : Indicate the cause of a FP exception
Flags bits : Indicate the raised FP exception for which the corresponding exception was not taken (disabled by the corresponding enable bit)
0 : Must be written 0, returns 0 on read
E : Unimplemented Operation　　O : Overflow
V : Invalid Operation　　U : Underflow
Z : Divide by 0　　I : Inexact

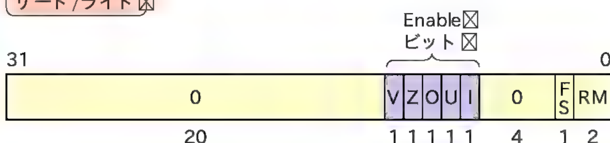
（d）FEXRレジスタ



FCC : Fp Condition Codes (used in conditional branches and moves)
FS : Flush to 0 (if set, denormalized inputs flushed to 0, denormalized results are flushed to 0 MinNorm (depending on rounding mode and result) instead to cause an unimplemented operation exception)
RM : Rounding Mode
: 0 - round to nearest　　: 1 - round toward zero
: 2 - round toward plus infinity　　: 3 - round toward minus infinity
0 : Must be written 0, returns 0 on read
Impl : Implementation dependent

E : Unimplemented Operation　　O : Overflow
V : Invalid Operation　　U : Underflow
Z : Divide by 0　　I : Inexact

（b）FCSRレジスタ



Enable bits : Enable bits that control if an exception is raised or just the corresponding flag is set on a FP exception
FS : Flush to 0 (if set, denormalized results are flushed to 0)
RM : Rounding Mode
: 0 - round to nearest
: 1 - round toward zero
: 2 - round toward plus infinity
: 3 - round toward minus infinity
0 : Must be written 0, returns 0 on read

E : Unimplemented Operation　　O : Overflow
V : Invalid Operation　　U : Underflow
Z : Divide by 0　　I : Inexact

（e）FENRレジスタ

図4　CP1を制御する際に使用するレジスタの構成

点を保持可能となります．

図3（p.57）に浮動小数点演算ユニット（CP1）に関連するレジスタ群を示します．また図4にCP1を制御する際に使用するレジスタの構成を示します．

4　システム・コプロセッサ0（CP0）の規定について

メモリ管理ユニットの制御方法，例外および割り込み処理方法，さらにマイクロプロセッサ・コアの制御，ステータス・レジスタ群は図5に示すように，CP0レジスタ群として規定されています．

● CP0メモリ管理ユニットについて

メモリ管理ユニットが提供する基本機能は，ソフトウェアで扱う仮想アドレスを，実際の物理アドレスに変換する機能，プログラム（タスク）ごとにセキュリティ機能をもたせること，また各仮想アドレス空間ごとに，そのアドレス空間のアクセス属性を定義付けるしくみとなります．

MIPS32アーキテクチャにおいては，図6に示す基本的に二つの特権レベル“ユーザ・モード”（CPUレジスタ群をアクセスでき，フラットな2Gバイトのアドレス空間をアクセス可能），および“カーネル・モード”（すべてのCPUリソースをアクセス可能で，4Gバイトのアドレス空間をアクセス可能）を規定しています．これらのモードの切り替えは，図6（a）に示すように，CP0のStatusレジスタのUMビット，EXLビット，ERLビットの組み合わせで行われます．

MIPS64アーキテクチャにおいては，図6（b）に示す三つの特権レベル“ユーザ・モード”，“スーパバイザ・モード”（ユーザ・モードおよびスーパバイザ・モードのリソースをアクセス可能），および“カーネル・モード”を規定しています．これらのモードの切り替えは，図6（b）に示すように，CP0のStatus

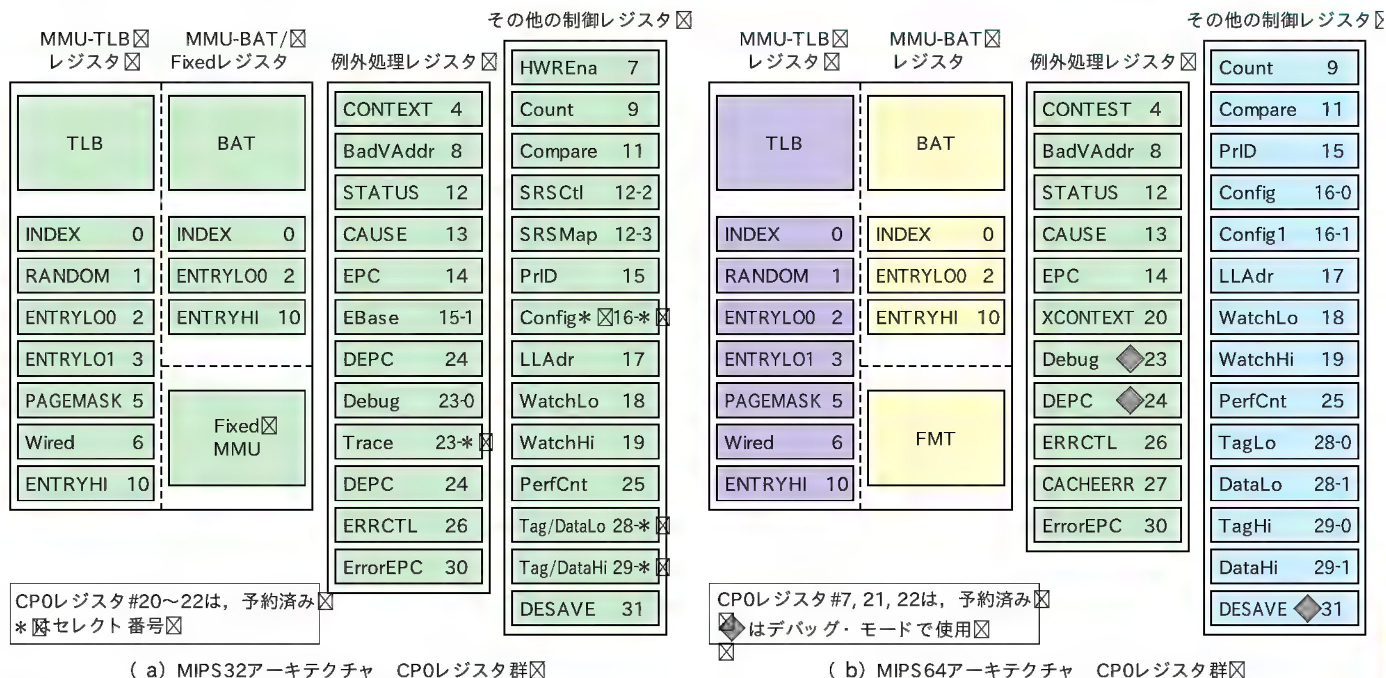


図5 システム・コプロセッサ0(CP0)の構成

これら一連のレジスタ群が，マイクロプロセッサのメモリ管理方法，例外・割り込み処理方法および細かなマイクロプロセッサ制御などを司どる

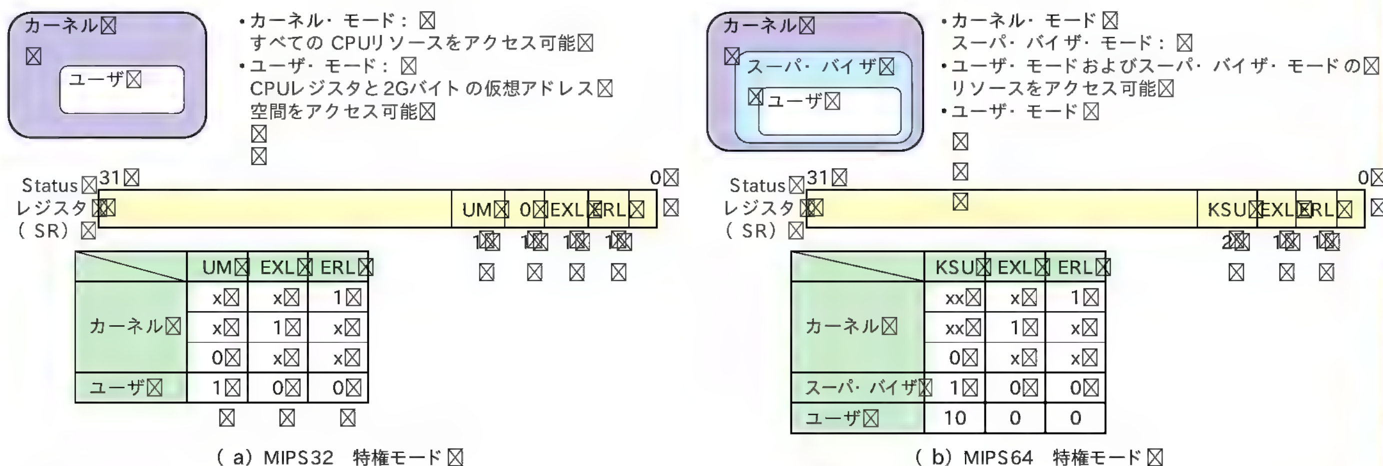


図6 特権モード

レジスタのKSUビット，EXLビット，ERLビットの組み合わせで行われます．

上記の基本モードに加えて，Debugモードが規定されており，MIPS32/MIPS64アーキテクチャともにそれらをインプリメントすることが可能です．また，MIPS32アーキテクチャにおいても，スーパ・バイザ・モードをインプリメントすることも可能です．

▶メモリ・マップ

MIPS32/MIPS64アーキテクチャにおいては，仮想アドレスの上位ビットを用いて，32ビット・アドレス・モード時，4Gバイトの物理アドレス空間を複数のセグメントに分けて管理します．64ビット・アドレス・モードのときは，物理アドレス空間は64Gバイトとなります．

これらのセグメントをアクセスする際は，プログラムはすべて仮想アドレスで行い，その仮想アドレスを物理アドレスに変換するユニットがメモリ・マネジメント・ユニット(MMU)です．MIPS32/MIPS64アーキテクチャでは，MMUの実装方式として，TLB機構，BAT機構，および固定(Fixed) MMU機構の三つを定義しています．

TLB機構は文字どおり，完全なTLBを搭載し，MIPS32およびMIPS64アーキテクチャで規定している仮想アドレス・セグメントのすべてを実装する必要があります．

31: 29) 仮想アドレス=(28: 0)

セグメント・セレクト

VA for TLB

4KEプロセッサ

仮想アドレス

TLB

32ビット

物理アドレス

32ビット

外部デバイス

(a) 仮想アドレスから物理アドレスへの変換概念

仮想アドレス

0xFFFF_FFFF	アドレス・エラー
0xE000_0000	
0xC000_0000	
0xA000_0000	
0x8000_0000	ユーザ・マップ・キャッシャブル useg
0x0000_0000	2Gバイト

ユーザ空間

仮想アドレス

0xFFFF_FFFF	カーネル・マップ・キャッシャブル kseg3 0.5Gバイト
0xE000_0000	カーネル・マップ・キャッシャブル kseg2 0.5Gバイト
0xC000_0000	カーネル・アンマップ・非キャッシュ kseg1 0.5Gバイト
0xA000_0000	カーネル・アンマップ・キャッシャブル kseg0 0.5Gバイト
0x8000_0000	ユーザ・マップ・キャッシャブル kseg 2Gバイト
0x0000_0000	

カーネル空間

(c) ユーザ・モード，およびカーネル・モードでの仮想アドレス・マップ

仮想アドレス

0xFFFF_FFFF	カーネル・マップ・キャッシャブル kseg3 0.5Gバイト
0xE000_0000	カーネル・マップ・キャッシャブル kseg2 0.5Gバイト
0xC000_0000	カーネル・アンマップ・非キャッシュ kseg1 0.5Gバイト
0xA000_0000	カーネル・アンマップ・キャッシャブル kseg0 0.5Gバイト
0x8000_0000	ユーザ・マップ・キャッシャブル useg/kseg 2Gバイト
0x0000_0000	

任意アドレス

任意アドレス

任意アドレス

物理アドレス

0xFFFF_FFFF

物理メモリ 3.5Gバイト

0x2000_0000

0.5Gバイト

0x0000_0000

注：物理アドレスは36ビット，ただしビット35～32は常時0

kuseg(2Gバイト)	MMUマップ	ユーザ領域とカーネル領域がオーバ・ラップした空間
kseg0(512Mバイト)	ダイレクト・マップ	一般的に，カーネルの実行モジュールやデータを配置する空間
kseg1(512Mバイト)	ダイレクト・マップ&非キャッシュ	一般的に，OSのI/Oレジスタ，ROMコードや，DISKバッファなどを配置する空間
kseg2(512Mバイト)	MMUマップ	スーパ・バイザ・シェルなどに許された空間
kseg3(512Mバイト)	MMUマップ	一般的に，OSのスタック，ユーザ・ページ・テーブル，ダイナミックに再配置されるデータなどを配置する空間

(b) 仮想アドレスの物理アドレスに対する割り付けマップ

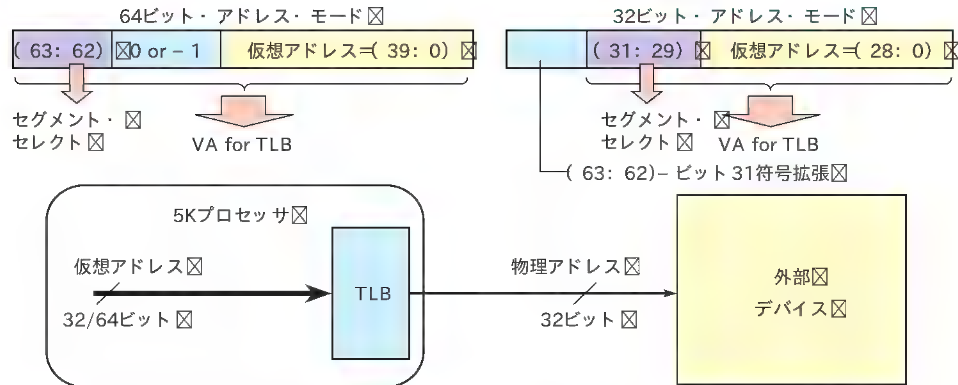
図7　MIPS32アーキテクチャにおけるメモリ・マップ

BAT機構は，簡易TLB方式のようなもので，ベース・アドレスにオフセットを加えて，物理アドレスを生成する手法です．この場合，kuseg，kseg0，kseg1は必ず実装しなければなりません．kseg2およびkseg3に関してはオプションとなります．

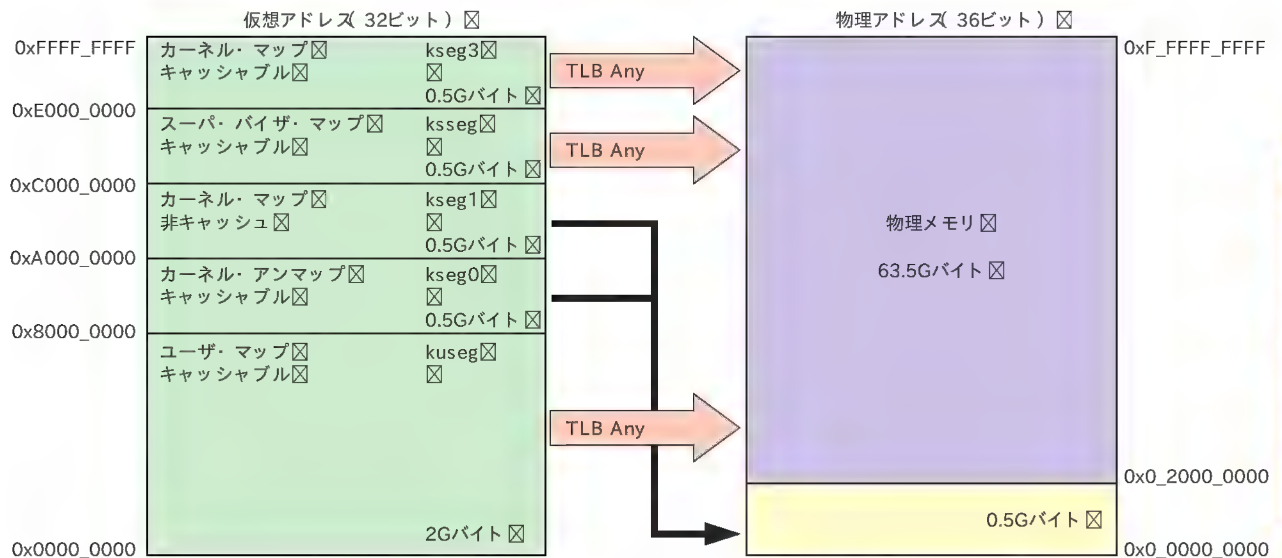
固定MMU機構は，ハードウェア的に固定化されたメモリ・マップを実現するもので，その場合は，kuseg，kseg0，kseg1は必ず実装しなければなりません．kseg2およびkseg3に関してはオプションとなります．

まずMIPS32アーキテクチャにおいて，図7(a)に仮想アドレスから物理アドレスへの変換概念，図7(b)に仮想アドレスの物理アドレスに対する割り付けマップ，図7(c)にユーザ・モードおよびカーネル・モードでの仮想アドレス・マップを示します．

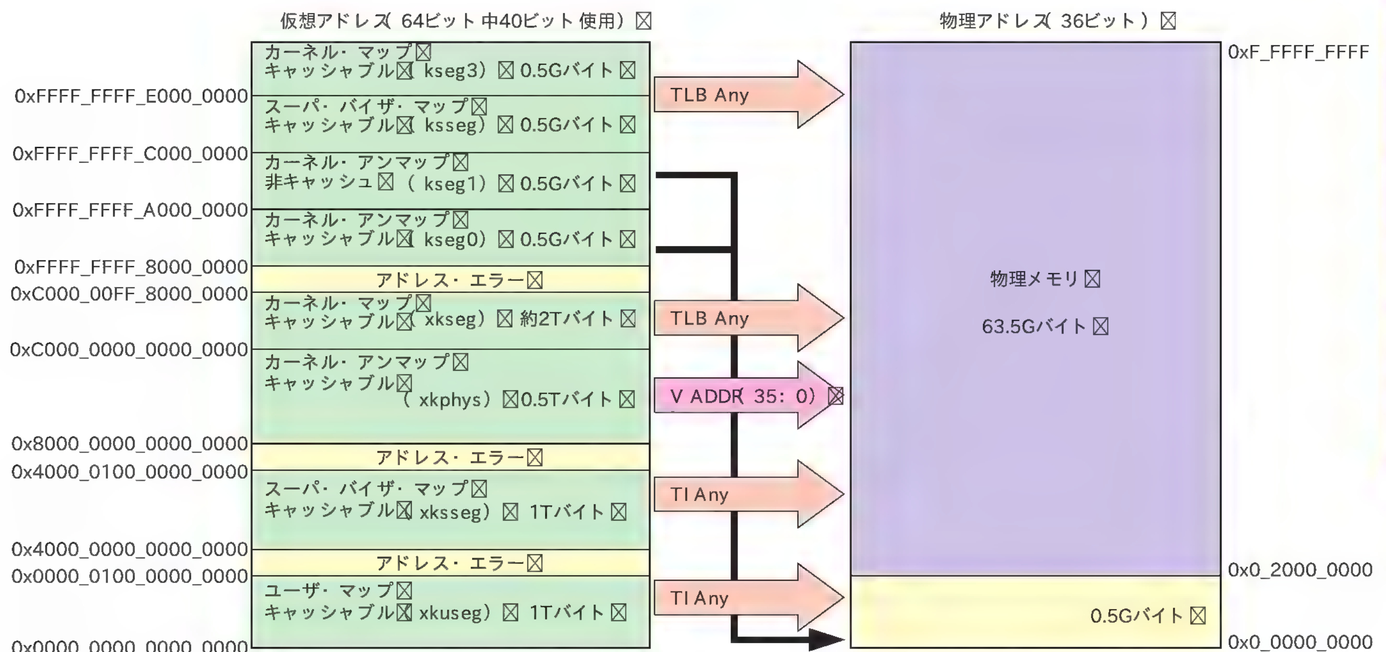
MIPS64アーキテクチャの場合，32ビット・アドレス・モードと64ビット・アドレス・モードが存在します．図8(a)に仮想アドレスから物理アドレスへの変換概念，図8(b)，(c)に仮想ア



(a) 仮想アドレスから物理アドレスへの変換概念

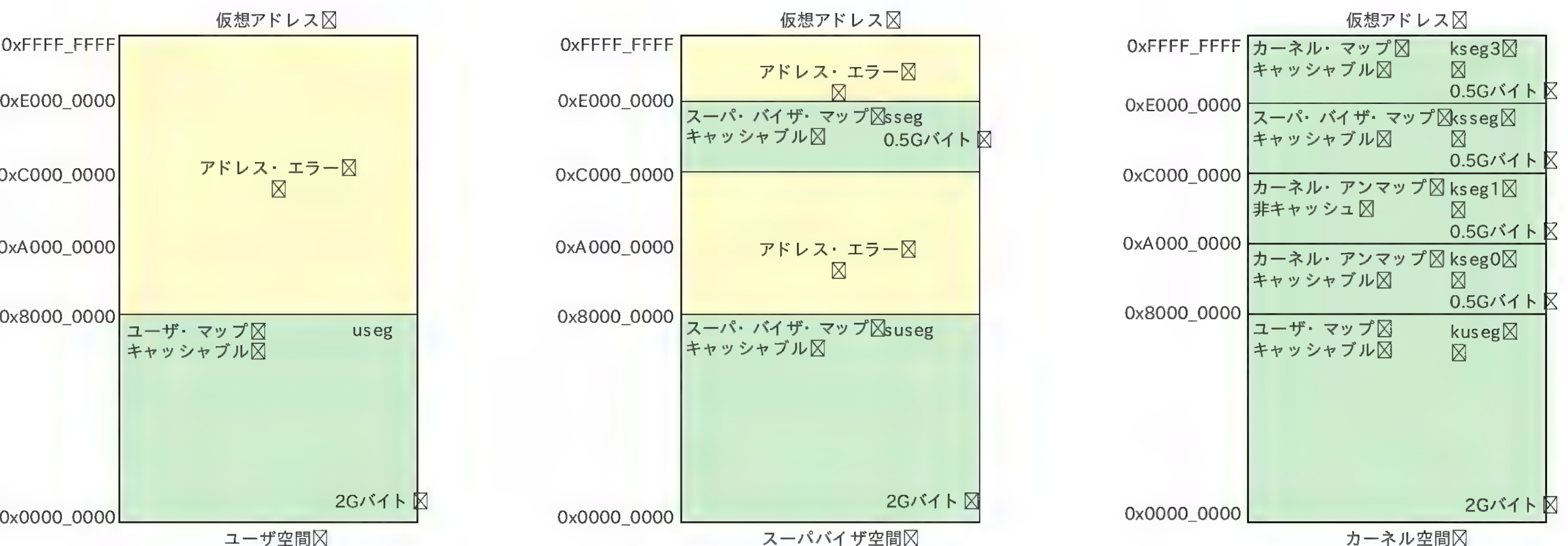


(b) 32ビット仮想アドレスの物理アドレス・マップに対する割り付け



(c) 64ビット仮想アドレスの物理アドレス・マップに対する割り付け

図8　MIPS64アーキテクチャにおけるメモリ・マップ



（d）32ビット・ユーザ・モード，スーパ・バイザ，およびカーネル・モードでの仮想アドレス・マップ

（e）64ビット・ユーザ・モードおよびカーネル・モードでの仮想アドレス・マップ

図8　MIPS64アーキテクチャにおけるメモリ・マップ（つづき）

ドレスの物理アドレスに対する割り付けマップ，図8(d)，(e)にユーザ・モードおよびカーネル・モードでの仮想アドレス・マップを示します．

▶TLB 構成

前述のメモリ・セグメントを実現するためにはTLBを実装する必要がありますが，MIPS32/MIPS64アーキテクチャでは，JTLB(Joint TLB)，ITLB(命令用)およびDTLB(データ用)を規定しています．これは仮想アドレス-物理アドレス変換を効率的に行うために，命令およびデータ共有のJTLBのほかに，命令用，データ用のITLBおよびDTLBをサブとして用意する手法です．JTLBのエントリ数に関しての規定はありませんが，最低限デュアル・エントリのフルTLBを実装する必要があり，ページ・サイズは，4Kバイトから256Mバイトをサポートしています(最小ページ・サイズは，1Kバイトが可能．ただしMIPS32/MIPS64アーキテクチャ規定では4Kバイトが最小)．各ページには，アドレス変換情報，ASID(プロセスID)およびキャッシュ・アクセス属性を保持する必要があり，TLBミスが発生した場合は，規定の例外を発生する必要があります．

ITLBおよびDTLBのエントリ数は四つで，各ページ・サイズは4Kバイトです．エントリの置換方式としてはLRU(Least Recently Used)方式を取ります．図9にTLB構成の概念図を示します．

● 例外処理，外部割り込み処理

例外処理および外部割り込み処理には，

1) 外部割り込み(システム状況に応じて発生)
2) 命令実行にともなったエラーで発生する例外，制御移行・トラップ例外，MMUに関連した条件による例外，パリティ・エラーなどのキャッシュに関わる例外(命令実行に応じて発生)
3) 特殊なウォッチ条件による例外，EJTAGデバッグによる例外(デバッグ・モードでの発生)

が挙げられます．基本的には，マイクロプロセッサの内部で発生する事象と外部で発生する事象があり，プロセッサの内部動作に完全に同期したものもあれば，非同期のものもあります．また，その実行サイクルは明示的なもの，予測不能なものと千差万別です．

図10は，例外の認識にともなって，パイプライン内の命令がどのように影響を受けるかを示しています．ここでは例として，MIPS32 4KEマイクロプロセッサ・コアの5段のパイプラインを示します．

▶ 例外，割り込みについて

マイクロプロセッサの内部で発生するさまざまな例外を表6に示します．図11にこれらの例外処理のマイクロプロセッサ・コア(MIPS32 4KEおよびMIPS64 5K)内の優先順位を示します．

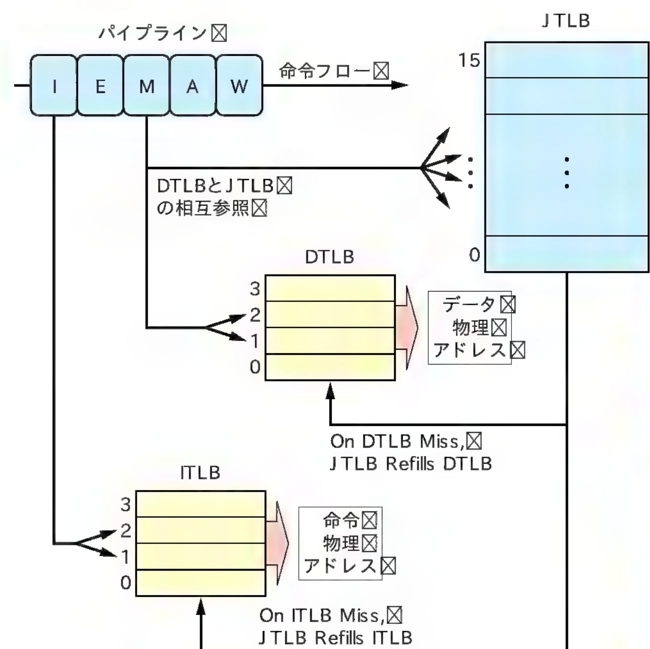


図9 TLB構成の概念図

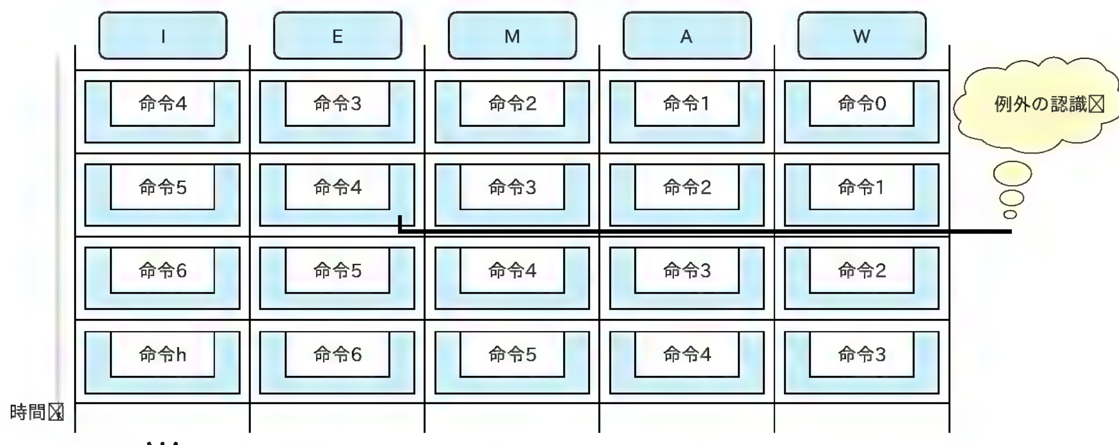


図10 例外認識とパイプライン動作

命令2および3に関しては，例外発生認識後も命令実行の完了まで継続される．しかし，命令4,5,6に関しては，命令そのものはパイプライン内を流れるが，命令の実際の実行は行われずに，キャンセルされる

Pr
1
2
Ap
Ap
3
4
5
6
Ap
Ap

表6　マイクロプロセッサの内部で発生するさまざまな例外

TLB Refill例外	命令フェッチもしくは，データ・アクセス時に発生 ●TLB miss：Statusレジスタの EXLビットが"0"で，TLBエントリにマッチするデータがない場合（MIPS32/MIPS64）
TLB Invalid例外	命令フェッチもしくは，データ・アクセス時に発生 ●TLB match：TLBエントリが Invalidの場合（MIPS32/MIPS64） ●TLB miss：Statusレジスタの EXLビットが"1"で，TLBエントリにマッチするデータがない場合（MIPS32/MIPS64）
TLB Modified例外	TLBストア動作時に，TLBエントリ内に Validなエントリがあるが，そのエントリが Dirtyである場合（MIPS32/MIPS64）
Bus Error例外（命令フェッチもしくは，データ・アクセス時に発生）	バス・タイム・アウト，無効な物理アドレスへのアクセス時に発生（外部回路にて通知される）（MIPS32/MIPS64）
Address Error例外（命令フェッチもしくは，データ・アクセス時に発生）	ハーフ・ワード境界に配置されていないデータに対するハーフ・ワードのロードもしくはストア動作時（MIPS32/MIPS64） ワード境界に配置されていないデータに対するロード，フェッチ，ストア動作時（MIPS32/MIPS64） ユーザ・モードからカーネル領域をアクセスした場合（MIPS32/MIPS64） ユーザ・モードからカーネル領域もしくはスーパ・バイザ領域をアクセスした場合（MIPS64）
Overflow例外	ADD命令/ADDI命令/SUB命令/SUBI命令などの結果オーバ・フローが発生した場合（MIPS32/MIPS64）
System Call例外	SYSCALL命令実行時（MIPS32/MIPS64）
Breakpoint例外	BREAK命令実行時（MIPS32/MIPS64）
予約命令例外	Majorオペコードもしくは Minorオペコードを持つ Special命令で，定義のないもしくは予約されている命令を実行した場合（MIPS32/MIPS64）． 32ビット・レジスタおよび動作モードにおいて64ビット動作を行った場合（MIPS64）
Trap例外	条件が成立して Trap命令が実行された場合（MIPS32/MIPS64）
コプロセッサ不正使用例外	コプロセッサが接続されていない状況で，コプロセッサ命令を実行した場合（MIPS32/MIPS64）． ユーザ・モードで，CP0命令を実行した場合（ユーザ・モード時に CP0命令が使用できない指定になっている場合）（MIPS32/MIPS64）
C2E例外	コプロセッサに一般例外を発生させるコプロセッサ2命令を実行した場合（MIPS32）
IS1例外	コプロセッサに implementation特殊例外1を発生させるコプロセッサ2命令を実行した場合（MIPS32）
IS2例外	コプロセッサに implementation特殊例外2を発生させるコプロセッサ2命令を実行した場合（MIPS32）
Machine Check例外	TLBエントリ内に，複数の Matchエントリ・データを確認した場合（MIPS32/MIPS64）
Watch例外（命令フェッチもしくは，データアクセス時に発生）	ソフトウェア・デバッグ時に，設定した Watch条件が成立した場合（MIPS32/MIPS64）
割り込み例外	規定されている八つの割り込み要因の一つが検知された場合（MIPS32/MIPS64）
RESET例外	パワーON時に，RESET信号がアサートされた場合（MIPS32/MIPS64）
Soft Reset例外	致命的エラー後システムが再起動し，Soft Reset信号がアサートされた場合（MIPS32/MIPS64）
Cache Error例外	命令もしくはデータ参照時にキャッシュ・パリティ・エラーが検知された場合（命令キャッシュ・パリティ・エラーに関しては一義的に処理サイクルが決定されるが，データ・キャッシュ・パリティ・エラーの場合は，ノンブロッキング・ロード機能との兼ね合いで処理サイクルが特定できない）（MIPS32/MIPS64）
デバッグ例外	EJTAGデバッグ例外（MIPS32/MIPS64） ●DINT：EJTAG Debug割り込みが発生した場合．EJ_DINT信号がアサートされた場合．ECRレジスタの EjtagBrkビットがセットされた場合 ●DIB：EJTAGのハードウェア・ブレーク命令条件が成立した場合 ●DDBS/DDBL：EJTAGデータ・アドレス・ブレーク条件が成立した場合，もしくは，データ値（アドレス+データ値）のブレーク条件が成立した場合 ●DBp：EJTAG Breakpoint（SDBBP命令を実行した場合） ●DSS：EJTAGのシングル・ステップ実行を行った場合

表7　MIPS32 4KEの割り込みモード設定

Statusレジスタ BEVビット	Causeレジスタ IVビット	IntCtlレジスタ VSビット	Config3レジスタ VINTビット	Config3レジスタ VEICビット	割り込みモード
1	x	x	x	x	互換割り込みモード
x	0	x	x	x	互換割り込みモード
x	x	0	x	x	互換割り込みモード
0	1	0	1	0	ベクタ割り込みモード
0	1	0	x	1	外部割り込みコントローラ・モード
0	1	0	0	0	不許可：IntCtlVS can not be non-zero if neither Vectored Interrupt nor External Interrupt Controller mode is implemented

MIPS32/MIPS64アーキテクチャでは，

1）互換割り込みモード

MIPS32およびMIPS64 Release1アーキテクチャ対応

2）ベクタ割り込みモード(IVモード)

優先順位化されたベクタ割り込み方式で，割り込み処理にシャドウ・レジスタ群を使用することが可能

3）外部割り込みコントローラ・モード(EICモード)

外部割り込みコントローラを用いた場合に優先順位化されたベクタ割り込み処理方法を規定したモード

の三つの割り込みモードが規定されています．表7に，MIPS32 4KEを例にそれぞれの割り込みモード設定を示します．

●互換割り込みモード

互換割り込みモード(ソフトウェアおよびハードウェア割り込み)では，二つのマスク可能なソフトウェア割り込みと内部パイプラインに対して非同期のマスク可能なハードウェア割り込み6本とNMIが一つ規定されています．ハードウェア割り込みは，レベル・センス方式で，アクティブ・ハイとなっており，NMIに関しては，エッジ・センスとなっています．これらのハードウェア割り込みは，マイクロプロセッサ・コアによりラッチされないので，それぞれの割り込み信号は，ソフトウェアによって対応する割り込み処理終了の通知があるまで，ハードウェアは保持する必要があります．また，それぞれのハードウェア割り込み処理の優先順位はソフトウェアにより規定する必要があります．

●ベクタ割り込みモード(IVモード)

ベクタ割り込みモード(IVモード)では，優先順位付けされたベクタ割り込みが規定されており，二つのマスク可能なソフトウェア割り込みとマスク可能な6本のハードウェア割り込みを処理可能です．それぞれのベクタ空間割り当ては，IntCtlレジスタのVSフィールドで規定し，汎用シャドウ・レジスタを割り当てることが可能です．図12にIVモードの構成概要を示します．

●外部割り込みコントローラ・モード(EICモード)

外部割り込みコントローラ・モード(EICモード)は，外部割り込みコントローラを用いた優先付けされた割り込み処理方法を規定しています．このモードの指定は，Config3レジスタのVEICビットで行われ，汎用シャドウ・レジスタを各割り込みに割り当てることが可能です．図13にEICモードの構成概要を示します．

▶ 例外処理・割り込み処理に関わるCP0レジスタ群

図14に例外処理および割り込み処理に関わるCP0レジスタ群を示します．

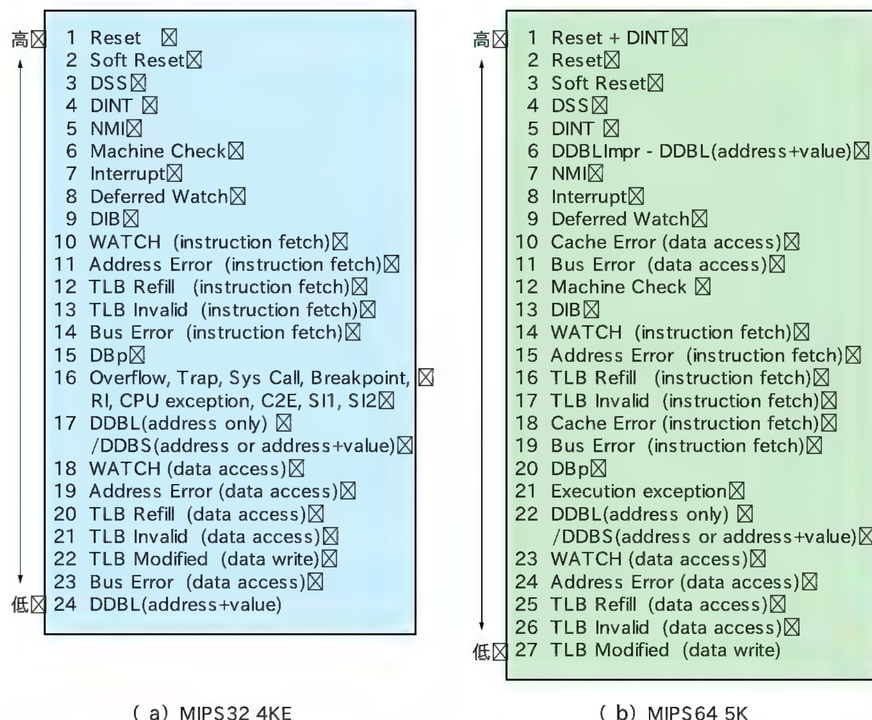


図11 例外処理時のマイクロプロセッサ・コア内の優先順位

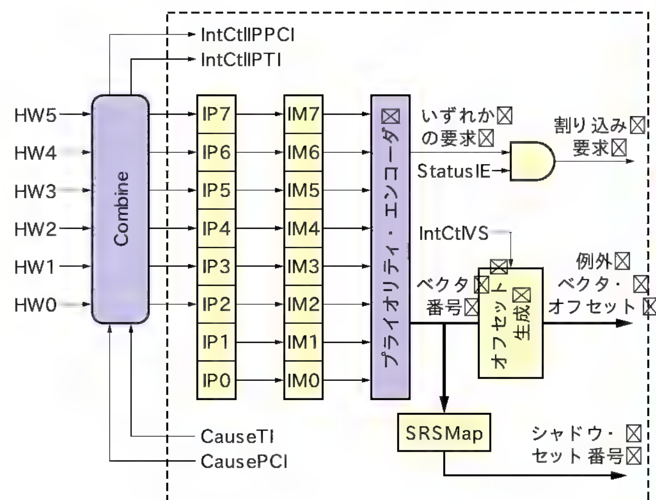


図12 IVモードの構成概要

群を示します．

CP0のStatusレジスタは，マイクロプロセッサ・コアのさまざまなステータスおよび動作モードを規定します．Statusレジスタの BEVビットおよびCauseレジスタのIVビットの組み合わせによって2種類の例外ベクタ・アドレスを選ぶことが可能です．

CP0のCauseレジスタは，さまざまな例外・割り込みの発生原因の情報を保持しています．

CP0のIntCtlレジスタ CP0#12Sel1は，ベクタ割り込みおよび外部割り込みコントローラをサポートするものです．

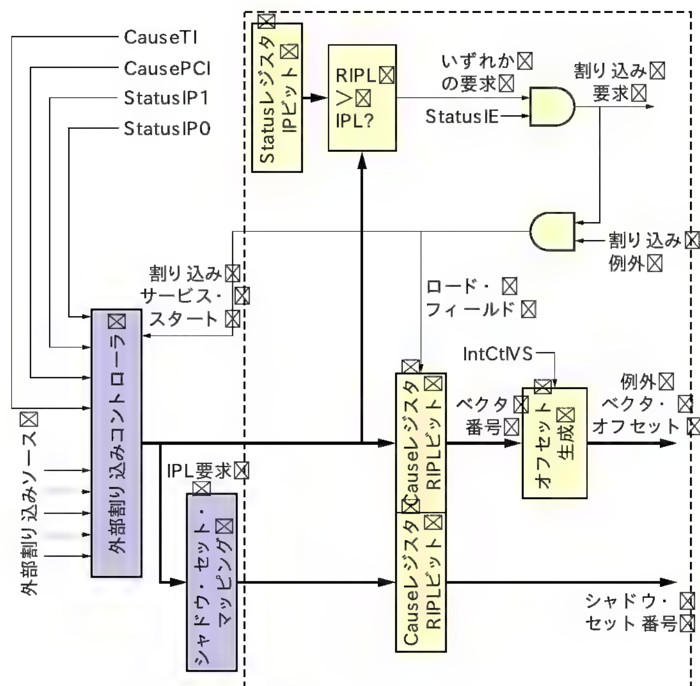


図13　EICモードの構成概要

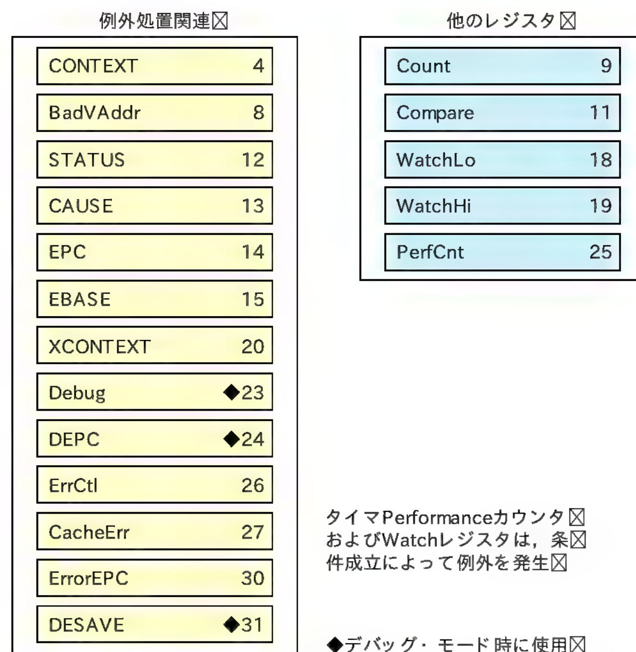


図14　例外処理および割り込み処理に関わるCP0レジスタ群

CP0のEBaseレジスタは，Statusレジスタの BEVビット＝"0"の場合，例外ベクタのベース・アドレスを保持しています．また，マルチプロセッサ・システムにおける個別のマイクロプロセッサ識別子も保持しています．

CP0のCacheErrorレジスタは，キャッシュ・エラーの発生原因などを保持しています．

CP0のEPCレジスタは，例外が発生した命令の仮想アドレスもしくは，Causeレジスタの BDビット＝"1"の場合は，直前で実行した分岐・ジャンプ命令の仮想アドレスを保持しています．Statusレジスタの EXLビット＝"1"の場合，本レジスタへの書き込みはできません．

CP0のErrorEPCレジスタは，リセット，ソフトウェア・リセット，NMIおよびキャッシュ・エラーの際に用いられ，例外処理後に復帰する命令に対する仮想アドレスを保持します．

CP0のBadVAddrレジスタは，AdEL，AdES，TLB Refill，TLB Invalid，TLB Modified例外などで発生するアドレス・エラーの際にその仮想アドレスを保持します．

CP0の32ビット・モード時のContextレジスタですが，このレジスタに格納される情報はBadVAddrレジスタの内容の一部ですが，このデータを用いて，OSはPTE（ページ・テーブル・エントリ）の参照を行います．64ビット・モード時は，XContextレジスタが使用されます．

CP0のCountレジスタは，マイクロプロセッサの動作クロックに合わせてカウントされるフリー・ラン・カウンタです．機能チェック，システム同期などに使用されます．

CP0のCompareレジスタは，Countレジスタに対して，トリガ値を設定するためのレジスタで，Countレジスタ値とCompareレジスタ値が等しくなると，Causeレジスタの IPxビットがセットされます．

Watchdogカウンタ用のWatchLoレジスタ，およびWatch/Hiレジスタは，これらのレジスタに設定した仮想アドレスに対して，リードもしくはライト動作が発生した場合に例外が発生します．

CP0のDebugレジスタは，デバッグ例外処理を行うためのデバッグ例外の原因などを保持します．非デバッグ・モードで，このレジスタに書き込みを行うとその状態は保障されません．また，デバッグ例外が発生した場合，DSS，DBp，DDBL，DDBS，DIB，DINT，HALT，DOZEおよびDBDフィールドは自動的にアップデートされます．非デバッグ・モードで，デバッグ例外が発生した場合，DSS，DBp，DDBL，DDBS，DIB，DINT，DExcCodeおよびDBDフィールドは自動的にアップデートされます．DebugレジスタのDExcCodeは，例外コードと同様です．

CP0のDEPCレジスタは，デバッグ例外処理後に復帰すべき命令に対する仮想アドレスを保持します．

● その他の例外

▶ デバッグ・ハンドラ・スクラッチパッド・レジスタ

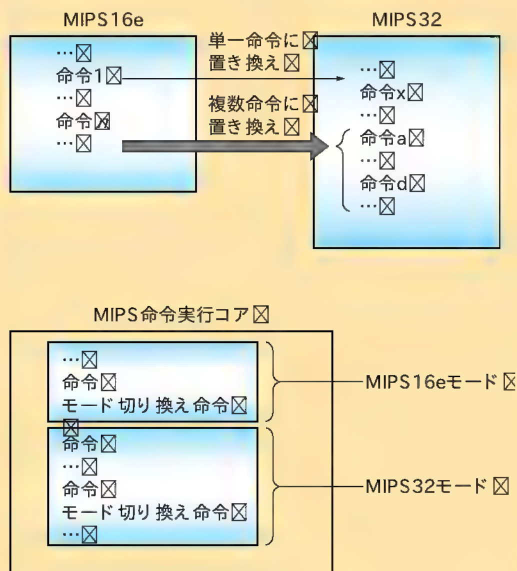
図15にCP0のDESAVEレジスタを示します．このレジスタが，"デバッグ・ハンドラ・スクラッチパッド・レジスタ"と呼ばれ，EJTAGプローブなど，あらかじめ指定されたメモリ空間に汎用レジスタの内容を保存するために用いられます．おもに例外ハンドラそのものをデバッグする際などに使用されます．

Information —— nVIDIA，TCP/IP および iSCSI オフロード・サービスの IReady 社を買収
米nVIDIA Corporation社は，TCP/IPおよびiSCSIオフロード・サービスを提供している米IReady社を買収したと発表した．

COLUMN 1

MIPS16e アプリケーション・スペシフィック・エクステンション(ASE)

組み込みアプリケーションにおいて，プログラムの容量をいかに小さくするかという命題は，今も昔も変わりません．MIPSアーキテクチャでは，コード・サイズの圧縮を行うために，Application Specific Extension(ASE)として，MIPS16e ASEを規定しています．MIPS16e命令セットは，固定16ビット長で，MIPS32もしくはMIPS64命令セットとペアで使用することができます．最大で40%程度のコード圧縮を行うことが可能です．



図A　MIPS16e命令の動作概要

ほとんどのMIPS16e命令は，単一のMIPS32命令に置き換えられるが，いつくかは複数のMIPS32命令の組み合わせに置き換えされたり，命令によってはMIPS16e命令固有のハードウェアに割り当てられる．
同一のメモリ空間にMIPS16eコードとMIPS32コードを配置することが可能で，それぞれのモードは，モード切り替えの命令を実行して行われる

コード・サイズの圧縮のメリットとしては，

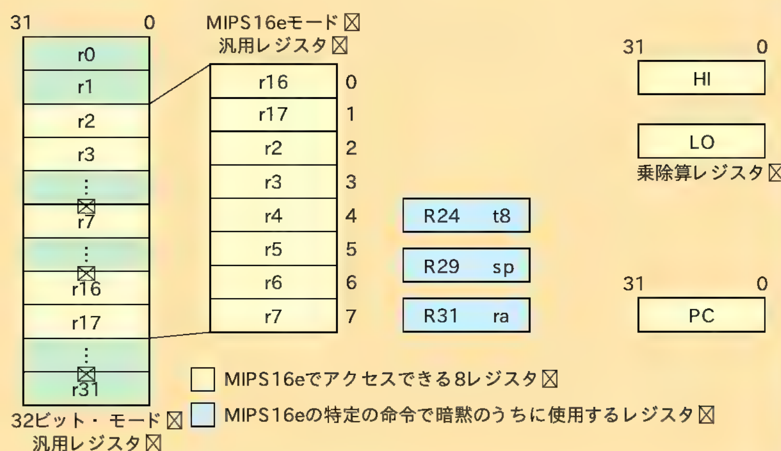
1) システム全体のメモリ容量を少なくすることにより，コストが削減できる
2) 同一サイズの命令キャッシュにより多くの命令を格納できるので，命令キャッシュのヒット率を向上させることができる
3) 単位時間当たりに実行できる命令数を増やすことができる

などが挙げられ，単にコスト削減のみならず，処理能力の向上を図ることも可能となります．

図AにMIPS16e命令の動作概要を示します．ほとんどのMIPS16e命令は，単一のMIPS32命令に置き換えることが可能で，Config1レジスタのCAビットの値によって，MIPS16e命令セットをサポートしているかどうかの確認を行うことが可能です．

図BにMIPS16eモードで使用されるレジスタ・セットを示します．MIPS16eモードでは，8本の32ビット汎用レジスタが使用されます．MIPS16eのMOVE命令は，32本すべての汎用レジスタをアクセスすることが可能です．ただし，R24は，t8として，R29は，spとして，R31は，raとして使用されます．

MIPS16e命令でサポートする命令の種類としては，ロード/ストア命令，Save/Restore命令，算術演算命令，ジャンプ/分岐命令，スペシャル命令となります．



図B　MIPS16eモードで使用されるレジスタ・セット

▶ パワーONリセット時

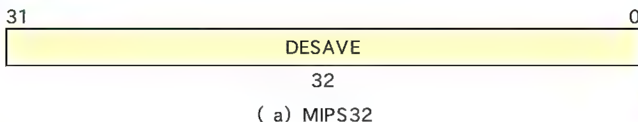
マイクロプロセッサのパワーON時に，リセット信号がアサートされることによりリセット例外が発生します．その際の仮想アドレスは，64ビット・モード時は0xFFFF_FFFF_BFC0_0000，32ビット・モード時は0xBFC0_0000で，アンマップでかつ非キャッシュ領域のアクセス(キャッシュおよびTLBはバイパスされる)から動作を開始します．

キャッシュ，TLBおよび表8のレジスタを除いて，内部データは不定となります．

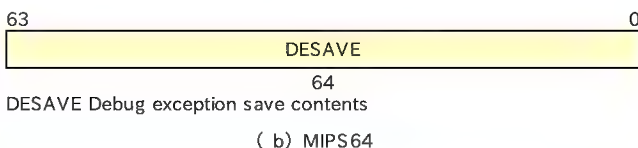
▶ ソフトウェア・リセット時

ソフトウェア・リセット信号がアサートされることによりソ

リード専用
オプショナル　コンプライアンス・レベル



(a) MIPS32



DESAVE Debug exception save contents

(b) MIPS64

図15　CP0のDESAVEレジスタ

表8 パワーONリセット時/ソフトウェア・リセット時/NMI発生時にセットされるレジスタ

- Randomレジスタ(最大値)
- Wiredレジスタ(0)
- Configレジスタ(Bootステートにセット)
- Stausレジスタ[BEV: 1(bootstrap vector locations)]
- TS: 0 SR: 0 NMI: 0(NMI例外は1')
- ERL: 1(exception level, kernel mode)
- ErrorEPCレジスタ(PCもしくはPC－4 if during branch delay)]
- PC = 0xBFC00000(64ビット・モードの場合は, PC = 0xFFFF_FFFF_BFC0_0000)

フトウェア・リセット例外が発生します．その際の仮想アドレスは，64ビット・モード時は0xFFFF_FFFF_BFC0_0000，32ビット・モード時は0xBFC0_0000で，アンマップでかつ非キャッシュ領域のアクセス(キャッシュおよびTLBはバイパスされる)から動作を開始します．

キャッシュ，TLBの内部データは不定となります．そのほかのレジスタの内容に関しては，表8のレジスタを除いて保持されます．

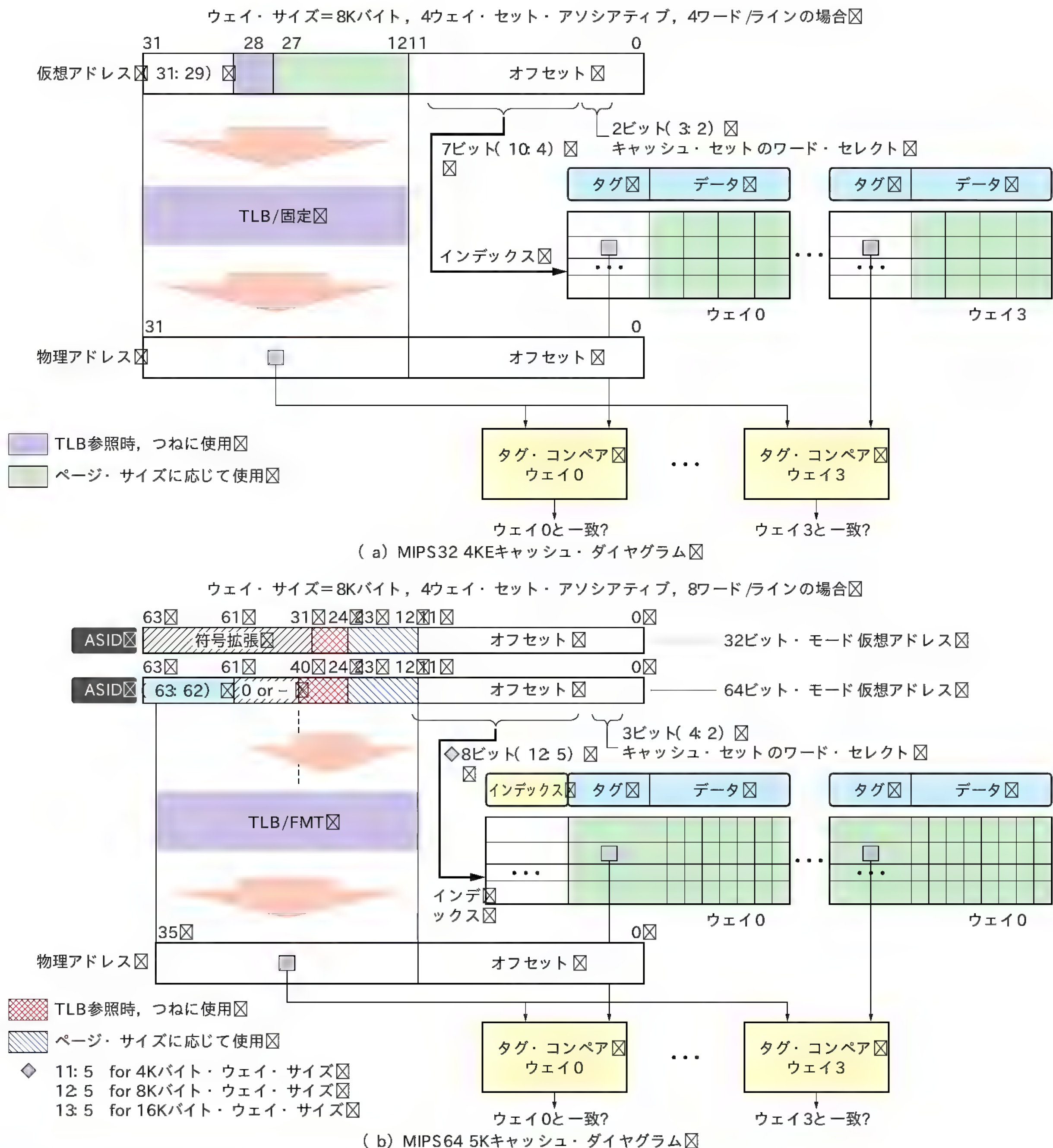


図16 キャッシュのブロック図

▶NMI 発生時

NMI 信号がアサートされることにより NMI 例外が発生します．この例外は一般的に復旧不可能な深刻なシステム・エラーが発生した場合，システムの再立ち上げを行う際に用いられます．その際の仮想アドレスは，64ビット・モード時は0xFFFF_FFFF_BFC0_0000，32ビット・モード時は0xBFC0_0000で，アンマップでかつ非キャッシュ領域のアクセス（キャッシュおよびTLBはバイパスされる）から動作を開始します．

キャッシュ，TLB の内部データ，そのほかのレジスタの内容に関しては，表8のレジスタを除いて保持されます（NMI ビットは'1'にセットされる）．

● キャッシュ構成および制御

マイクロプロセッサ・コアにおいて，キャッシュは主記憶から内部パイプラインへ高速に命令およびデータを供給するバッファとして位置づけられます．MIPS32/MIPS64アーキテクチャでは，その構造を次のように規定しています．

- キャッシュのアソシアティビティ
  ダイレクト・マップ方式（1ウェイ）から最大8ウェイまでを規定
- キャッシュのタグ情報
  仮想タグ・アドレスもしくは物理タグ・アドレスをサポート
- キャッシュのインデックス情報
  仮想インデックスもしくは物理インデックスをサポート
- キャッシュのライト・ポリシ
  ライト・スルー/ノー・ライト・アロケート，ライト・スルー/ライト・アロケート，ライトバック方式をサポート
- キャッシュ・ライン・サイズ
  4，8，16，32，64，128バイトをサポート
- ウェイごとのキャッシュ・ライン数
  0，64，128，256，512，1024，4096サポート
- 搭載可能なキャッシュ容量
  アソシアティビティ X ライン・サイズ
  X ウェイごとのライン数

メモリ管理ユニットの項で説明したように，仮想アドレス空間をセグメントに分割して管理し，それぞれの仮想アドレス空間は物理アドレス空間に割り付けられています．その割り付けの際に，その領域の情報をキャッシュ・メモリを踏まえてどのように取り扱うかを規定しているのが，"キャッシュ属性"もしくは"キャッシュ・アクセス属性"となります．Config レジスタのCもしくはK0ビット・フィールドなどで規定されます．キャッシュ構成の例として，図16（a）に MIPS32 4KE のキャッシュのブロック図，図16（b）に MIPS64 5K のキャッシュのブロック図を示します．

▶キャッシュ管理に関連する CP0レジスタ

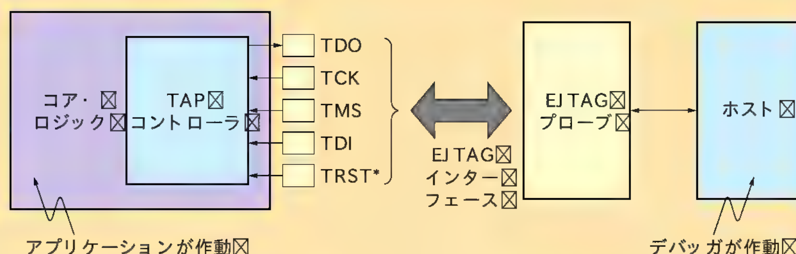
キャッシュ管理を行ううえで必要となる CP0レジスタとしては，キャッシュ内のタグ・アレイにデータを書き込むために用

COLUMN 2

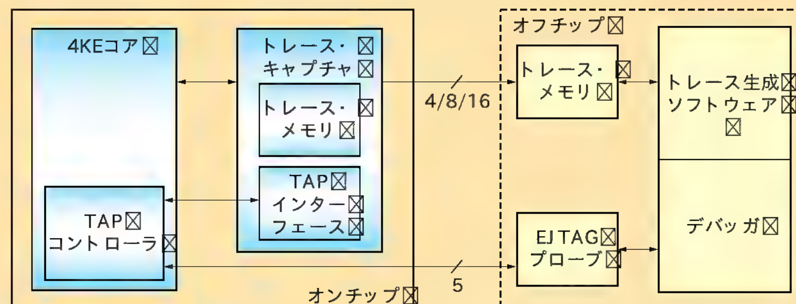
EJTAG　デバッグ機能

MIPS 系のマイクロ・プロセッサでは，デバッグ機能として EJTAGデバッグ仕様を規定しています．EJTAGデバッグ機能では，TAPを介してターゲット・チップおよび EJTAG機能の認識を行います．基本的には，命令のシングル・ステップ実行，ソフトウェア・ブレーク・ポイントのサポート，仮想アドレスやデータ値を用いたハードウェア・ブレーク・ポイントのサポートを行っています．プロセッサは，シリアル・インターフェースを介して EJTAG空間へのリードおよびライトを行います．デバッグにおいて NMI および外部割り込みもサポートしています．図Cに EJTAGデバッグ機能の概要を示します．

また，PDtrace機能をコアに搭載することにより，リアルタイムで命令およびデータのトレースを EJTAGを介してサポートすることも可能です．図Dに PDtrace機能の概要を示します．



図C　EJTAGデバッグ機能の概要



図D　PDtrace機能の概要

トレース・メモリは，オンチップに搭載した場合，標準的な EJTAG プローブを介してアクセス可能．オフチップの場合は Trace 用のピンを用いてアクセスする

いられるTagLoレジスタ，TagHiレジスタ，キャッシュ内のデータ・アレイからデータを読むために用いられるDataLoレジスタ/DataHiレジスタ，およびキャッシュ内のテストに用いられるErrCtlレジスタがあります．キャッシュを管理する場合，おもにCACHE命令を用いて行います．おもな処理としては，キャッシュ・フラッシュ，キャッシュ・ロック，キャッシュ・フィル，およびキャッシュ読み出し(キャッシュ・テストのため)になります．

まとめ

今後もMIPS Technologiesでは，基本アーキテクチャの拡充・改良を推し進め，将来の組み込みアプリケーションで必要となる仕様を特定アプリケーション拡張(ASE)として規定していきます．また，Appendix1で解説する32ビットMIPS32 24Kコア・アーキテクチャは，前述したさまざまな機構を実装することにより，"高い柔軟性・自由度"，"プログラマビリティ性の向上"，そして"スケーラブルな高い処理能力"を提供できるようにし，さらにこれらの基本アーキテクチャおよびASEを実装した，フル・シンセサイザブル・コア製品を展開して行きます．

現在，一般的なSoCは，さまざまなハードウェアIPブロックの集積により構成されています．しかし一方では，システムで必要となる機能ブロックを，より積極的にソフトウェアで実装し，一端開発を終了したSoCの製品寿命を延ばすとともに，さまざまな組み込みアプリケーションへ適応可能なSoCが求められています．また，増大する一途の90nmおよび将来の最先端プロセスをベースとした設計開発費，マスク費用を効率的にカバーし，収益率を向上させることも視野に入れる必要があります．MIPS Technologiesは，これらの需要に対応可能な新しい32ビット・プロセッサのフル・シンセサイザブル・コア群を充実させていくつもりです．

参照文献

(1) Stephen B. Furber著，豊橋技術科学大学 今井正治 監訳，"比較研究 RISCアーキテクチャ VLSI RISC Architecture and Organization，基礎から学ぶ，プロセッサ設計とVLSIチップの実例"，日経BP社
(2) MIPS32 4KEアーキテクチャ・インテグレーション，Training Class資料
(3) MIPS64 5Kアーキテクチャ・インテグレーション，Training Class資料

なかがみ・かずふみ　ミップス・テクロノジーズ

第1部 Appendix 1

0.13μmプロセスで400MHz～550MHz動作を実現するソフト・コア

フル・シンセサイザブル・コアMIPS32 24Kの概要



中上 一史

1998年以降，MIPS Technologies社では，MIPS32/MIPS64アーキテクチャを実装したさまざまなフル・シンセサイザブル（ソフト）・コアおよび一部ハード・コアを設計開発してきました．表Aは，今日現在のコア製品群を示しています．それぞれのコアの詳細に関しては，MIPSのWebサイトの各のデータシート，インテグレータ・ガイドおよびソフトウェア・ガイドを参照してください．

● MIPS32 24Kの内部構造

図Aに24Kの内部ブロックの概要を示します．この24Kは，単一命令発行の8段パイプライン構造をもった32ビット・シンセサイザブル・コアで，TSMC社の0.13μmのGプロセスで400MHz（ワースト・ケース），0.13μmのLVプロセスで500MHz（ワースト・ケース），0.13μmのLV-ODプロセスで550MHzの動作周波数を達成します．

メモリ・サーブ・システムは，64ビット幅を採用し，アウト・オブ・オーダでノン・ブロッキング方式のバス・アクセスをサポートしています．昨今のSoC設計/開発において，システム・レベルのパフォーマンス・ボトルネックが，メモリ・アクセス，およびバス転送能力にあることが，一般的によく知られています．そのため，コアそのものは32ビットですが，コアのデータ・バス幅を64ビットにすることにより，そのボトルネックを少しでも解消できるようにしたものです．

メモリ管理方式（MMU）としては，さまざまなOSをサポートするために，完全なTLBを実装したMMU，もしくは簡易的なTLBをベースとするMMUを選択可能です．また，ユーザのシステム設計のノウハウを最大限に生かすために，ユーザ命令拡張機能（CorExtend機能）を搭載しています．

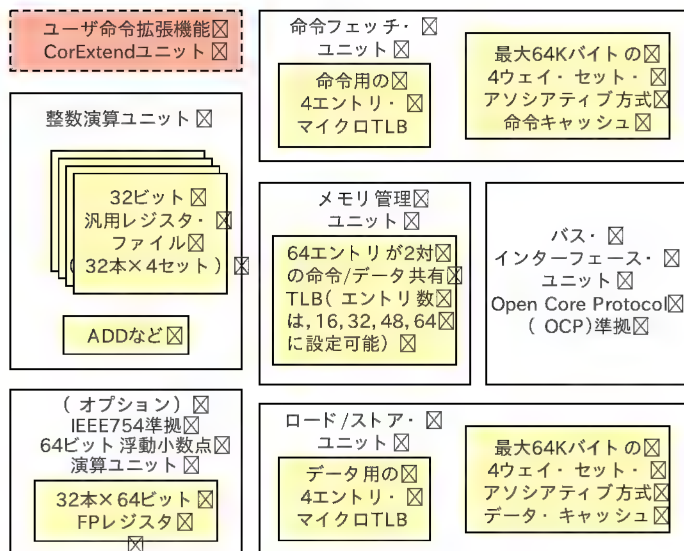
外部IPと24Kコアを接続するIPバス・プロトコルとして，Broadcom社，TI社，Nokia社という大手半導体/システム・メーカが提唱するOpen Core Protocol（OCP：http://www.ocpip.org/）を採用しました．

命令コード効率（コード・サイズ）を最大40%高めるMIPS16e ASE（Application Specific Extention）を搭載しています．また，コアの低消費電力化を行うために，コアに可能な限りのゲーテッド・クロック手法を活用しました．

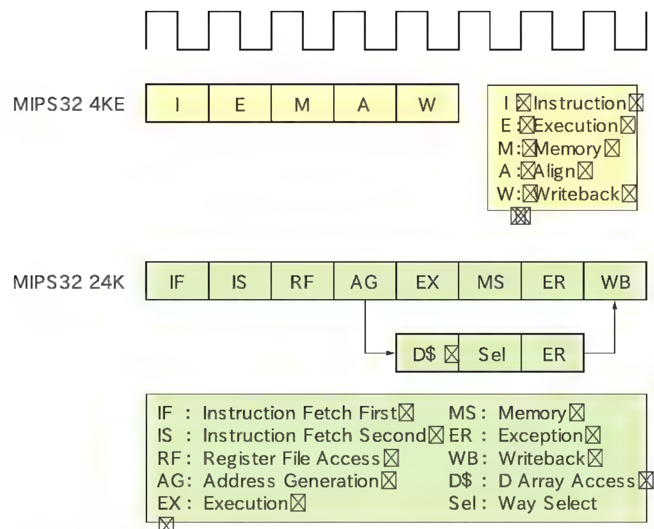
業界標準のMIPS32アーキテクチャを基本としているので，複数の大手ツール・ベンダより，さまざまな業界標準開発支援ツールが

表A フル・シンセサイザブル（ソフト）・コア一覧

実装アーキテクチャ	コア名	シンセサイザブル（ソフト）・コア	ハード・コア	ハード・コア対応プロセス	シンセサイザブル・コア動作周波数 プロセス・ルール0.18μm時（MHz）	プロセス・ルール0.13μm時（MHz）
MIPS32	4Kc	YES	YES	SMIC 0.18μm	160～180	210～255
MIPS32	4Km	YES	NO	―	160～180	210～255
MIPS32	4Kp	YES	NO	―	160～180	210～255
MIPS32 Release2	4KEc	YES	YES	TSMC 0.18μm	―	210～255
MIPS32 Release2	4KEm	YES	NO	―	―	210～255
MIPS32 Release2	4KEp	YES	NO	―	―	210～255
MIPS32 Release2	4KEc Pro	YES	NO	―	―	210～255
MIPS32 Release2	4KEm Pro	YES	NO	―	―	210～255
MIPS32 Release2	4KEp Pro	YES	NO	―	―	210～255
MIPS32 Release2	4KSc	YES	NO	―	―	200
MIPS32 Release2	4KSd	YES	NO	―	―	200
MIPS32 Release2	4KSc Pro	YES	NO	―	―	200
MIPS32 Release2	4KSd Pro	YES	NO	―	―	200
MIPS32 Release2	M4K	YES	NO	―	―	200～240
MIPS32 Release2	M4K Pro	YES	NO	―	―	200～240
MIPS32 Release2	24Kc	YES	NO	―	―	400～550
MIPS32 Release2	24Kf	YES	NO	―	―	400～550
MIPS32 Release2	24Kc Pro	YES	NO	―	―	400～550
MIPS32 Release2	24Kf Pro	YES	NO	―	―	400～550
MIPS64	5Kc	YES	NO	―	―	350
MIPS64	5Kf	YES	NO	―	―	320
MIPS64	20Kc	NO	YES	TSMC 0.13μm	―	533
MIPS64	25Kf	NO	YES	東芝 90nm	―	600～800



図A　MIPS32 24Kの内部ブロック



図B　MIPS32 24Kパイプライン概要

入手可能です．アプリケーションのデバッグ，プロファイリングには，MIPS系標準のEJTAGを活用しています．

● MIPS32 24Kパイプラインとは

図Bは，従来のMIPS32 4KEパイプラインと，MIPS32 24Kパイプラインを比較したものです．この図が示すように，24Kでは将来にわたって，高い処理能力と高速な動作周波数を容易に実現できるように，パイプライン段数を5段から8段に変更し，かつ整数演算パイプラインと，ロード/ストア・パイプラインを分離しました．以下では，将来の処理能力と動作周波数に対するスケーラビリティを考慮して24Kに実装した個別の機能について概説します．

▶ 命令フェッチ

命令フェッチは，1stステージと2ndステージの二つで構成され，命令キャッシュより連続する2命令をフェッチします．24Kの命令キャッシュは，最大64Kバイトで，4ウェイ・セット・アソシアティブ方式で構成されており，比較的遅いメモリ・セルでも，キャッシュ・メモリを構成しやすいよう，2サイクル・キャッシュ・アクセスを前提としています．オプションで，パリティ保護を設定でき，クリティカルな命令列を，つねにキャッシュに常駐できるように，ラインごとにロックする機能も搭載しています．

従来のコアでは，キャッシュの構成に関して，ダイレクト・マップ，2ウェイ，4ウェイが選択可能になっていましたが，キャッシュのヒット率，ダイ資源の効率的利用の観点から4ウェイ固定を選択しました．また，昨今のSoC設計においてもっとも難しいのがキャッシュ・メモリの実装ですが，キャッシュ・アクセスに2サイクルを許すことにより，コアの動作周波数に対して，比較的遅いメモリ技術を用いても，最適なコア性能を達成できるようにくふうしました．

仮想アドレスから物理アドレスへの変換はTLBを用いて行いますが，TLBミスのペナルティを削減するためために，命令・データ共有のTLB以外に命令/データ用の4エントリのマイクロTLBを実装しました．さらに，命令の実行部分と，命令フェッチ部分を完全に独立させ，デコード済み命令を格納する6エントリの命令バッファを設けました．これにより，さまざまなミスをともなう処理に依存して，命令実行が妨げられることがなくなりました．

パイプラインの流れを乱す要因の一つである分岐動作にともなうペナルティを削減するために，512エントリ分岐履歴テーブルを持つダイナミック分岐予測機構を実装し，4エントリのリターン・スタックを設けました．また，命令キャッシュ・ミスは，同時に二つの処理を可能とし，MIPS32命令デコーダとは別に，独立したMIPS16e命令用デコーダを設けることにより，それぞれの命令の効率的なデコードを可能にしました．

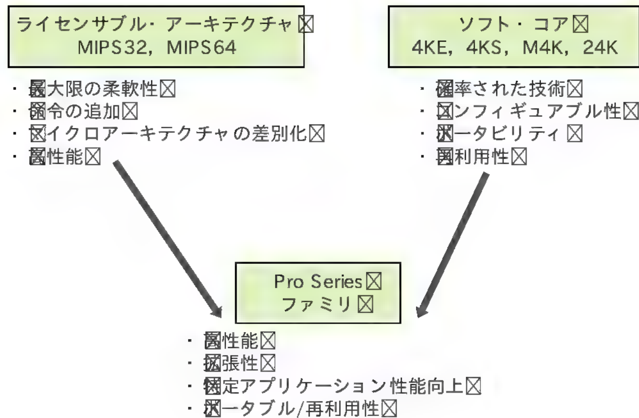
▶ 命令実行

プロセス技術の進歩にともなって，将来的に高い動作周波数を容易に実現できるようなパイプライン構造を念頭にパイプラインの各段を最適化し，整数演算パイプラインと，ロード/ストア・パイプラインを分離すること（スカラ化）で，実行パイプラインがキャッシュ・ミスなどで妨げられる要因を排除しました．

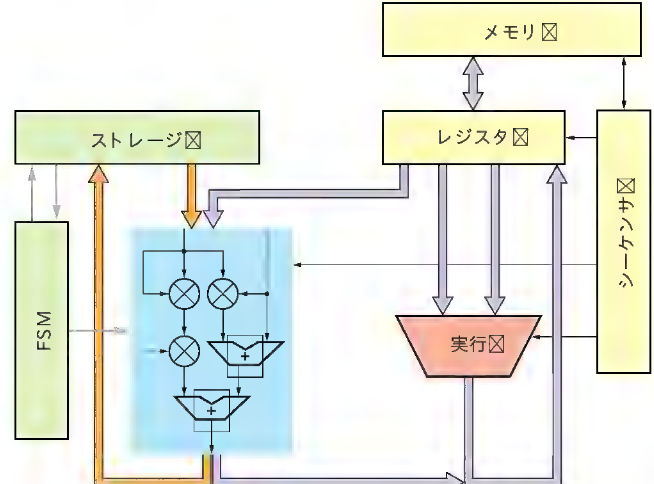
24Kでは，オプションで浮動小数点ユニットを提供します．この浮動小数点パイプラインも整数演算パイプライン，ロード/ストア・パイプラインと同時に実行可能で，浮動小数点演算パフォーマンスとしては，1MFLOPS/MHzで，内部に32本の64ビット浮動小数点レジスタを搭載しています（MIPS32 Release2アーキテクチャで規定した32ビット・マイクロプロセッサに，64ビット・コプロセッサを接続した初の実装例）．演算ユニットは完全にクロックに同期して動作し，演算結果を次の演算に渡すバイパス機能を有します．32×32→64ビットの掛け算を，5サイクルで実行可能です．

▶ ロード/ストア部

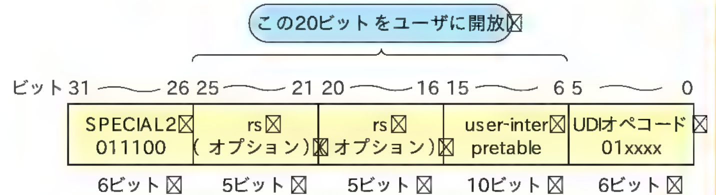
24Kのデータ・キャッシュは最大64Kバイトで，4ウェイ・セット・アソシアティブ方式，32バイト/ラインで構成されており，オプションで，バイト・パリティ保護を設定可能で，クリティカルなデータ列を，つねにキャッシュに常駐できるように，ラインごとでロックする機能も有しています．将来の処理能力向上を容易に行うために，1次データ・キャッシュのタグ・メモリ部に，スヌープ用のポートを個別に用意しました．もちろん，このスヌープ機構は，2



図C　Proシリーズ



図D　CorExtendの概念



図E　基本UDI(User Defined Instruction)フォーマット

次キャッシュをもった24Kの派生マイクロプロセッサ・コアへの布石でもあります．

ロード・ストアにともなう四つのデータ・キャッシュ・ミスを処理できます．アプリケーション・プログラムから，pre-fetch命令などを用いて，明示的にデータ・キャッシュ内のデータを制御することが可能で，I/Oなどへのデータの書き込みをサポートするために，Critical word forwarding動作をサポートしています．

● Proシリーズ

業界標準のマイクロプロセッサ・コアをSoC設計に応用するメリットは，その汎用性(設計開発ツールの豊富さ，豊富な多数のOSサポート，過去の設計資産の有効活用，そして，将来に対する継続性など)にありますが，その一方で，各半導体メーカおよびシステム設計メーカは，自社製品においていかに競合他社に対して，差別化要素を明確にして，そのSoCデバイスの市場寿命を延長し，かつ市場価値を向上させることができるかに躍起になっています．

その一つのアプローチ方法が，昨今，話題となっているリコンフィギュアブル・プロセッサですが，この場合，現在入手可能なものでは，前述した標準のメリットを最大限に生かすことはできません．

従来，MIPS社では，自社の持てる技術を結集して，その独自性・差別化要素を最大限に実現する方法として，MIPS32アーキテクチャおよびMIPS64アーキテクチャのライセンス提供を行ってきました．このアーキテクチャ・ラインセンスを取得することにより，ライセンシは，MIPSアーキテクチャに基づいた32ビット，64ビットのマイクロプロセッサを独自に開発することができます．

しかし，これを行うには，豊富な資金力，豊富で優秀なマイクロプロセッサの設計・開発能力，そして多大な熱意が必要になるので，ほとんどの場合，大手半導体メーカがそれを行ってきたわけです．

しかし，前述のリコンフィギュアブル・プロセッサに対して，その可能性の高さに注目しているのは，半導体メーカより，むしろシステム設計メーカです．そこでMIPS社は，従来提供してきた高いリコンフィギュアビリティを持つフル・シンセサイザブル・コアに，新たな機能として，CorExtend機能を開発し，それを実装したコアをProシリーズとして製品化しました(図C)．

▶ ユーザ命令拡張機能CorExtend

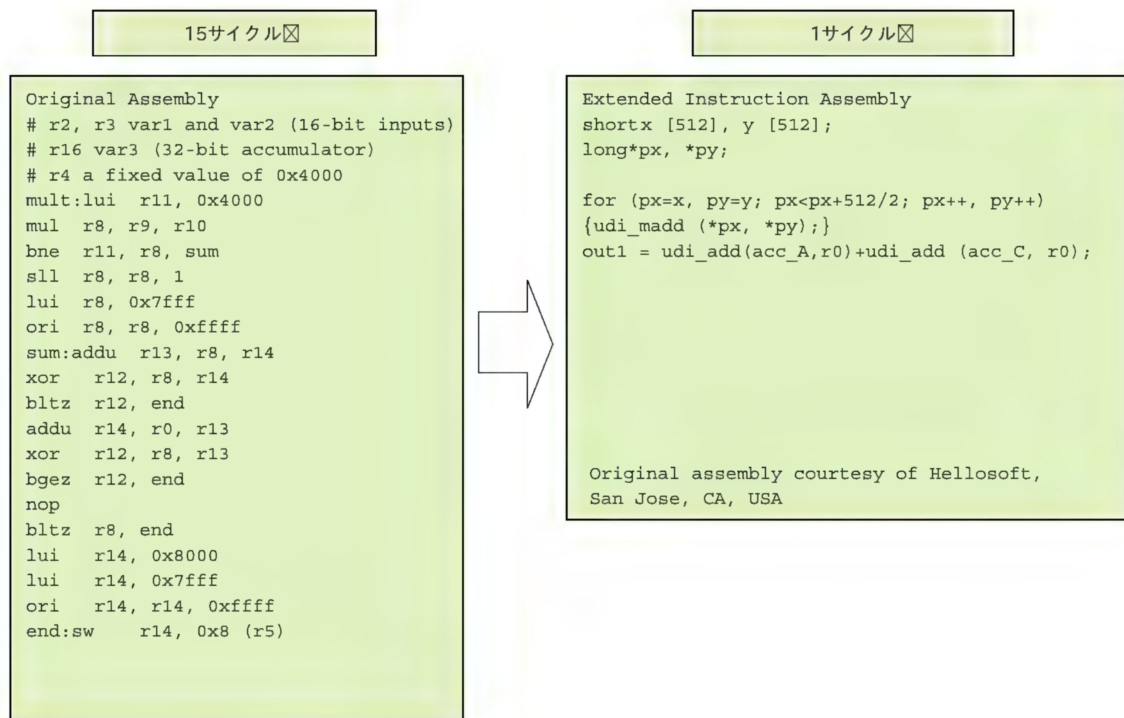
アプリケーション・ソフトウェアの開発においても，その処理能力の向上を図る際に，開発段階でソフトウェアのプロファイリングを行い，全体の実行時間に占めるクリティカル・ルーチンを洗い出し，まずそのクリティカル・ルーチンの最適化を行う手法が一般的に取られてきました．また，システム設計メーカでは，自社の独自技術を実現するために，このソフトウェアとハードウェアを組み合わせて，特定用途に特化したハードウェア・ブロックを開発し，独立したデバイスとして応用し，さらに一歩進んで，そのハードウェア・ブロックをほかの機能ブロックと組み合わせて，ASIC化するといったアプローチを採ってきました．

しかしその延長線上で，つねにコストの問題(ダイ・サイズの増加)，開発費の問題(マスク・コストの増加)，発熱の問題(パッケージ，コストの増加)など，さまざまな複合的な問題に直面しています．

そこで，古くて新しいこの問題に，MIPS Technologies社では従来から提唱してきた"高いプログラマビリティの実現"をさらに一歩進め，16個の命令追加をできるしかけを作りました．それがCorExtend機能です．

図Dに CorExtendの概念を示します．前述したようなユーザが持つハードウェア機能ブロックをコアに最適化した形で取り込み，それの機能ブロックをソフトウェアから，命令レベルで積極的に活用して，そのアプリケーション・レベルにおけるパフォーマンスを最大限に得ようという発想から来ています．

図Eに基本UDI(User Defined Instruction)フォーマットを示します．20ビットをユーザに開放しているので，このビットを自由に用いて，その命令の動作を規定することが可能になっています．ここで重要な点は，アーキテクチャの互換性を厳格に保つために，16



図F　VoIPの処理の一部

個の命令オペコードは，MIPS社が決めている点にあります．これにより，汎用のCコンパイラでその命令をサポートするコードを生成することができます．

▶UDIによる高速化例

この CorExtend 機能を活用して，VoIPの処理の一部をユーザ命令として規定して実行した場合と，オリジナルのMIPS32命令で実行した場合との比較を図Fに示します．オリジナルのコードでは15サイクル費やしていた処理を，UDIを定義することにより，この例の場合，1サイクルに縮めることができます．つまり15倍の高速化を実現したことになります．

この詳細なインプリメントの過程に関しては，MIPS社のWebサイトからアプリケーション・ノートなどをダウンロードすることができるので，そちらを参照してください．

なかがみ・かずふみ　ミップス・テクノロジーズ

第1部 Appendix 2

Windowsコマンド・プロンプト上で動作する

MIPSアーキテクチャ・エミュレータsimips



中森 章

はじめに

本稿では，MIPSアーキテクチャの動作を理解するための助けとなるように，ソフトウェアによるMIPSエミュレータを作成してみた．何事も，習うより慣れろ，百聞は一見に如かずの精神である．

最初に作成したバージョンはほかのところでも発表したが，それに数々のバグ修正と機能拡張を加えたものを今回発表する．今回の目玉は，MIPS64 Release2の命令対応とMIPS-3Dへの対応である．

本エミュレータは次号付属CD-ROMや本誌のWebサイトから，エミュレータ本体のソースやサンプル・プログラム付きでダウンロードできる．ソースも添付しているので，Linuxなどほかの OSにも移植できるだろう．

● エミュレーション・プロセッサの仕様

本エミュレータは次のMIPSプロセッサをエミュレーションする．

- 命令セット： MIPS64(Release2)，MIPS16，MIPS16e，MIPS-3D
- 命令キャッシュ： 32Kバイト，ダイレクト・マップ形式，ライン・サイズ16バイト，パリティなし
- データ・キャッシュ： 16Kバイト，ダイレクト・マップ形式，ライン・サイズ16バイト，パリティなし
- 仮想アドレス64ビット，物理アドレス32ビット
- ダブル・エントリ形式TLB，48エントリ
- リトル・エンディアンに対応．リバース・エンディアン(ユーザ・モードでのエンディアン反転)には非対応
- EJTAG関連の命令やレジスタには非対応
- パフォーマンス・カウンタ関係の命令には非対応
- Release2の命令に対応しているが，割り込み時のレジスタ・バンクには非対応

● エミュレータの機能

本エミュレータは次のような機能をエミュレーションする．

- プログラムのロード/実行/トレース
- レジスタのダンプ/更新
- メモリ内容のダンプ/更新
- アセンブル
- 逆アセンブル
- 命令コードまたはアドレスを指定してのブレーク
- コンソール出力/入力

入出力機能としては，コンソールの入出力機能を実装している．物理アドレスの`0x0F010000`に書き込みを行うと，コンソール出力として書き込まれた1バイトの値をASCIIコードとして文字を表示する．逆に`0x0F010020`から読み出しを行うとコンソール入力としてキーボードから，`0x0F010040`から読み出しを行うとファイルから1バイトのデータを入力する．

なお，このコンソール入出力用のアドレス以外の空間は，RAMがマップされていると考えてよい．

● ロード可能なプログラムの形式

ロード可能なプログラムの形式は次のとおり．

- ELF実行形式
- インテルHEX形式
- モトローラHEX(Sレコード)形式
- アセンブラ・ソース形式
- メモリ・イメージ形式(16進数で記述されたアドレスとデータの組)

このように，一般的なMIPSのクロス・コンパイル環境でコードを生成すれば，そのまま本エミュレータでもロード可能なファイルとなる．ただし実機を想定したコンパイラでは，スタートアップ・ルーチンやプラットホーム依存のコードが生成される場合があるので，そのあたりを注意して使う必要がある．

筆者が使用しているCコンパイラは，GCC3.2と，トランジスタ技術2002年8月号の付録CD-ROMに収録されている，Cygwin上で動作するリトル・エンディアン専用のGCCコンパイラmipsel-linux-gccなどである．しかし，それぞれレジスタの使い方に多少違いがあったり，使用できないコンパイル・オプションなどがあるので，詳細は本エミュレータの説明書を参照してほしい．

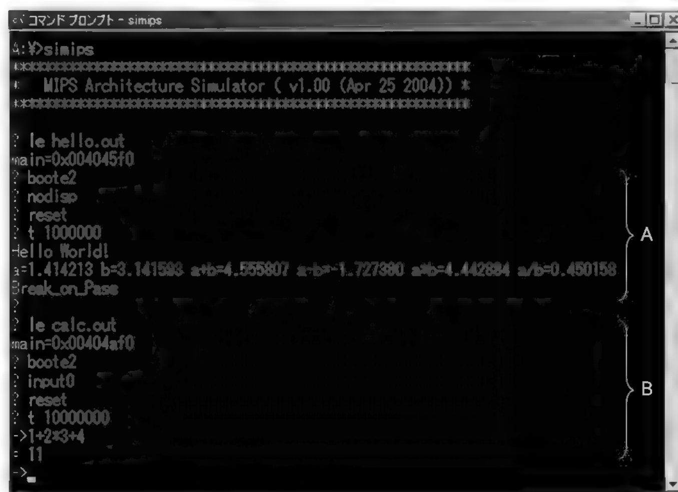
また本エミュレータのコンソール入出力をプログラムから簡単に使えるように`printf`や`scanf`関数群も用意しており，それらを使えば容易にコンソール入出力が可能である．

● エミュレータ実行例

本エミュレータの操作コマンドを表A(p.76)に示す．

エミュレータを起動するには，EXEファイルのアイコンを直接ダブルクリックしたり，コマンド・プロンプトを開き，

```
>simips
```



図A　simipsの動作のようす

表A　エミュレータ操作コマンド一覧（おもなもの）

q	エミュレータの終了
reset	プロセッサにリセットをかける．具体的にはPCが0xbfc00000に設定され，ステータス・レジスタのERLビットがセットされるだけ．MIPSの仕様にはないが，キャッシュやTLBも無効化する
t 回数	PCが示すアドレスから命令をトレースする．回数を省略すると1命令のみをトレースする．回数にinfを指定すると，CTRL-Cが押されるかブレーク条件が成立するまでトレースを続ける
disp ユニット	tコマンド実行時にレジスタの値をダンプするユニットを指定する．ユニットにはcpu，cp0，fpuが指定できる．allを指定するとすべてのユニットのレジスタをダンプする．instを指定すると逆アセンブル・コードを表示する
nodisp ユニット	dispコマンドの逆．tコマンド実行時にレジスタをダンプしないユニットを指定する．既定値は， disp inst disp cpu nodisp cp0 nodisp fpu である
a アドレス	ライン・アセンブラ（文法はエミュレータ添付の説明を参照）．アドレスは仮想アドレスを想定しているが，下位29ビットを物理アドレスとみなして，その物理アドレスの位置にアセンブルしたコードを格納する
a16 アドレス	ライン・アセンブラ．MIPS16/MIPS16e用
u 仮想アドレス 回数	逆アセンブラ．アドレスは仮想アドレスなのでマップ領域（アドレス変換を行う領域）を逆アセンブルする場合は，アドレス変換の例外が発生しないようにする必要がある．回数は省略できる．その場合は20を指定したとみなされる（20命令逆アセンブルする）
le ファイル名	ELF実行形式のプログラムのロード
li ファイル名 オフセット	インテルHEX形式のプログラムのロード．オフセットは省略可能（0が指定されたとみなす）
r ユニット	指定したユニット（cpu，cp0，fpu）のレジスタの現在の値をダンプする．
bi0 命令コード メッセージ	命令コードによるブレーク・ポイントの設定．メッセージにはブレーク・ポイントにマッチしたときに表示する単語（1語）を指定する．bi0のみを入力するとブレーク・ポイントの許可/禁止が切り替わる．初期値はbi0 0x03fffc0d Break_on_Passを指定したのと同等である．これはbreak 0xffff0命令である
bi1 命令コード メッセージ	命令コードによるブレーク・ポイントの設定．メッセージにはブレーク・ポイントにマッチしたときに表示する単語（1語）を指定する．bi1のみを入力するとブレーク・ポイントの許可／禁止が切り替わる．初期値はbi1 0x03ffffcd Break_on_Failを指定したのと同等である．これはbreak 0xfffff命令である
boot	リセット・アドレス（0xbfc00000）に li r1,0x80002000 jr r1 に対応する命令コードを書き込む．bootコマンドはサンプル・プログラムが0x80002000番地から始まることを想定している．
boote2	mips1/mips2のbootコードと例外ハンドラを生成する
boote3	mips3/mips4のbootコードと例外ハンドラを生成する
input0	I/Oの入力ポートを0x0f010020に設定する（規定値）．これはコンソールから対話的に文字を入力する場合に使用する
input1	I/Oの入力ポートを0x0f010040に設定する．これは__mips.inpというファイルの内容を入力データ列として使用する．バイナリ形式でデータを取り込むので，改行コードが0x0d，0x0aとなることに注意（MS-DOS互換テキスト・ファイルの場合）
m 物理アドレス データ	物理アドレスで指定されるアドレスに1ワード（32ビット）のデータを書き込む．
dw 物理アドレス1 物理アドレス2	物理アドレス1から物理アドレス2までの物理アドレスで指定されるメモリの内容をワード（32ビット）単位に表示する
dh 物理アドレス1 物理アドレス2	物理アドレス1から物理アドレス2までの物理アドレスで指定されるメモリの内容をハーフ・ワード（16ビット）単位に表示する
db 物理アドレス1 物理アドレス2	物理アドレス1から物理アドレス2までの物理アドレスで指定されるメモリの内容をバイト（8ビット）単位に表示する
h	ヘルプ・メッセージを表示する

注：コマンドに与えるアドレスや数値は10進数または16進数である．16進数の場合は，0x123cdのように，頭に「0x」を付加する．

と入力して起動する．起動すると“?”というプロンプトを表示してコマンド入力待ちとなる．

ここではまず，コンソールにメッセージを表示するサンプル・プログラム（hello.out）を実行してみる．次のようにコマンドを入力する（hello.outはカレント・ディレクトリにあるものとする）．

```
le  hello.out
boote2
nodisp
reset
t 1000000
```

すると，図A（p.71）のAのようにメッセージが表示される．

また，コンソール入力を使った例として，電卓プログラム（calc.out）も作成した．次のようにして起動する．

```
le calc.out
boote2
reset
input0
nodisp
t 10000000
```

すると“->”というプロンプトが出るので数式を入力する．

```
->1+2*3+4   （入力）
=  11       （出力）
->          （次の入力待ち）
```

というように計算結果が表示される（図AのBの部分）．何も入力せずリターンのみを入力した場合や，エラーがあった場合はプログラムを終了する．

なお，この電卓プログラムのコンソール入力は，キーボードから対話的にキーを入力できるので，input0を指定しておく．

＊　　＊

アーカイブ・ファイルには，本エミュレータのWindows版の実行形式ファイル以外に，エミュレータ本体のソース一式，そして各種コンパイラで作成したサンプル・プログラムも収録しておくので，参考にしてほしい．

なかもり・あきら

第2部 Chapter 3 GHzオーダのプロセッサを組み込みへ

PMC-Sierra RMシリーズの概要とRM7900 & RM9000x2GLの詳細

Paul Cobb/長島 資記

カラー・レーザ・プリンタや高速プリント複合機といった分野では，GHzオーダのプロセッサが要求される．本章で取り上げるPMC-Sierra社のRMシリーズは，この分野でのシェアが高いプロセッサである．SysADバスで接続するプロセッサ単体型から，メモリ・コントローラなどを内蔵した周辺コントローラ集積型など，ラインナップが揃っている．（編集部）

1 PMC-Sierra社とMIPSプロセッサの歴史

PMC-Sierra社は，現在MIPSアーキテクチャで最高レベルの性能のRISC CPUを量産/提供できるベンダです．PMC-Sierraといえば，もともとカナダの通信インフラ機器向け半導体製品の主要ベンダとして知られていましたが，2000年に米国のQED社を買収・統合したところから，MIPSプロセッサの主要ベンダの一つとして現在に至っています．

1984年にスタンフォード大学の研究成果を商業化したMIPS Computer Systems社の主要アーキテクト Tom Riordan氏とRay Kunita氏が，1991年にスピンアウトしてQuantum Effect Design（QED）を設立しました．QEDはR4700など高性能なMIPSプロセッサ・コアを設計し，NECやIDTといった主要なMIPSプロセッサ・ベンダに提供していました．そして1997年に社名をQuantum Effect Devicesに変更し，独自のCPU製品であるRM5200シリーズの提供を開始しました．

高性能を追求してきたQED/PMC-Sierra社は，現在は250MHzから1GHzを越える幅広い性能のCPUを提供しています．ますます高性能なCPUが必要なカラー・レーザ・プリンタや高速プリント複合機といったプリンティングの領域や，ディジタル・コンシューマ，またPMC-Sierraがもともと得意とする通信装置などの分野で広く採用され，最近はデータ・ストレージ，車載アプリケーション，工業制御アプリケーションと，さらに市場を広げ応用分野を増やしています．

2 製品のラインナップとロード・マップ

● ラインナップ概要

PMC-SierraのMIPSプロセッサは，おもにMIPS64アーキテクチャを採用しており，RM5200およびRM7000ファミリは，世界の64ビットRISCプロセッサ全体の出荷数量を牽引しています．これらの製品はとくにルータの制御プレーン，レーザ・プリンタの制御などの高性能なプロセッサが求められる組み込み機器市場においては，それぞれそのトップ・サプライヤの製品に採用されています．PMC-Sierraはその高性能MIPSベース製品をシームレスにシステム・アップできるよう，ピン・ソフト互換を確保しつつラインナップを整えています（表1）．

また，業界でもいち早く完全な鉛フリー・パッケージの供給を2003年より開始しており，すでに量産ベースで安定供給しています．

● プロセッサ単体型と周辺コントローラ集積型

PMC-SierraはQED当時より，つねに業界最速のMIPSベース・マイクロプロセッサを提供することを標語としています．MIPS標準のSysADバスを中心に，高いパフォーマンスを提供しながら，低消費電力と低価格化を実現してきました．基本はコアの高速化が第一です．現在の製品ラインナップは，独自機能やインターフェースを効率的に集積したシステム・コントローラASICに対する高速，低価格で拡張性あるCPUブロックとしてのプロセッサ単体型の製品と，周辺機能も取り込みシステム・レベルのソリューションを提供する周辺コントローラ集積型製品群の二つの方向性があります（図1，p.79）．

また，プロセッサ単体製品では，スムースなパフォーマンスの拡張を実現するために，低速から高速まで各コアを同一ピン配置のパッケージに封止し，一つのボード・デザインを幅広いパフォーマンスの製品に共通に活用できるよう，パッケージの選択に注意を払っています．中でもExposedPadというQFPに似たパッケージで，GND信号を外周のピンではなく底部にまとめて配置したパッケージは，RM5200A，RM7000C，RM7900の各ファミリに渡って広範囲のパフォーマンスをカバーするピン互換ソリューションとして採用されています．ピン数の削減とともに，熱をデバイス底部のPadからボードに

表1　PMC-Sierra MIPSベース・プロセッサ一覧

	RM5200A		RM7000C			RM7900		
製品	RM5231A	RM5261A	RM7000C	RM7035C	RM7065C	RM7900	RM7935	RM7965
動作周波数(MHz)	250, 300, 350, 400		466, 533, 600			668, 750, 835, 900		
内蔵キャッシュ	L1命令/データ：32Kバイト/32Kバイト		L1命令/データ：16Kバイト/16Kバイト L2：256Kバイト			L1命令/データ：16Kバイト/16Kバイト L2：256Kバイト		
外部キャッシュ制御	なし		64Mバイト	なし		64Mバイト	なし	
SysADバス幅	32ビット	64ビット	64ビット	32ビット	64/32ビット	64ビット	32ビット	64/32ビット
SysAD周波数	LVTTL 133MHz(400MHz)		LVTTL 133MHz, HSTL 200MHz			LVTTL 133MHz, HSTL 200MHz		
デバッグ	ROMエミュレータ・サポート		ROMエミュレータ・サポート			EJTAG/トレース・バッファ		
消費電力	1W / 400MHz		2W / 600MHz			4.7W / 900MHz		
パッケージ	128 QFP 128 ePad	208 QFP 216 ePad	304 TBGA	128 ePad	216 ePad 256 TBGA	304 TBGA	128 ePad	216 ePad 256 TBGA

(a)プロセッサ単体型

	RM9000		RM9000GL					
製品	RM9100	RM9200	RM9120	RM9122	RM9124	RM9220	RM9222	RM9224
動作周波数(MHz)	800, 1000		800, 1000					
内蔵キャッシュ(コア毎)	L1命令/データ：16Kバイト/16Kバイト，L2：256Kバイト		L1命令/データ：16Kバイト/16Kバイト L2：256Kバイト					
外部キャッシュ	なし		なし					
システム・バス幅	64ビット SysAD(200MHz)		64ビット SysAD(200MHz)					
メモリ・コントローラ	200MHz DDR-SDRAM		200MHz DDR-SDRAM(ECC)					
通信I/O	なし		DUART 10/100/1000 MAC×3	DUART 10/100/1000 MAC×2	DUART 10/100/1000 MAC×3 or (GPI-16×2+10/100/1000 MAC×1)	DUART 10/100/1000 MAC×3	DUART 10/100/1000 MAC×2	DUART 10/100/1000 MAC×3 or (GPI-16×2+10/100/1000 MAC×1)
システム・バス	SysAD HT ローカル・バス		SysAD HT ローカル・バス	PCI ローカル・バス	SysAD HT or PCI ローカル・バス	SysAD HT ローカル・バス	PCI ローカル・バス	SysAD HT or PCI ローカル・バス
その他機能	8Kバイト・スクラッチRAM		8Kバイト・スクラッチRAM					
デバッグ	EJTAG/トレース・バッファ		EJTAG/トレース・バッファ					
消費電力	5W/1GHz	10W/1GHz	―					12W(全I/O稼動時)
パッケージ	672 FCBGA		672 FCBGA		896 FCBGA	672 FCBGA		896 FCBGA

(b)周辺コントローラ集積型

放出することで，高い放熱効率を実現しました．これにより，高速でもファンレスのプロセッシング環境を提供し，組み込み機器での熱対策と高速化を同時に解決しています(図2)．パッケージには64ビットSysADバスの216ピンExposedPadのラインナップと，32ビットSysADバスの128ピンExposedPadのラインナップがあります．

●各ファミリの概要

▶RM5200Aファミリ

MIPS IV ISA互換の汎用64ビットMIPSプロセッサとして，1997年に発売されたRM5200ファミリの製品です．一つの整数演算と一つの浮動小数点演算を同時に実行できる，スーパ・スカラ・アーキテクチャで，400MHzで880MIPS(Dhrystone MIPS2.1：コンパイラとオプションによる：以下同じ)という性能を発揮します．外部バスの幅が32ビットのRM5231Aと64ビットのRM5261Aがあります．データと命令に32Kバイトずつの1次キャッシュを搭載し，2ウェイ・セット・アソシアティブで動作します．周辺機能やメモリにアクセスするためのSysADバスは，動作周波数により最高133MHzで動作し，250MHzから400MHzのコア動作周波数をサポートしています．

低価格・低消費電力で高パフォーマンスなRM5200Aファミリは，ディジタル・コンシューマ機器や，ルータなどの通信機器，プリンタなどにも多く採用され，ロング・セラーとなっています．

▶RM7000ファミリ

RM7000Cシリーズ(RM7000C，RM7035C，RM7065C)は0.13μmプロセスで，300MHzから600MHzという高い動作周波数を提供します．RM7000ファミリはMIPS IV ISA互換の独自CPUコアに，命令・データそれぞれ16Kバイトの1次キャッシュに加え，256Kバイトの2次キャッシュを搭載し，高いパフォーマンスを提供しているのが大きな特徴です．RM7000Cシリーズではシステム・バス(SysAD)の動作周波数

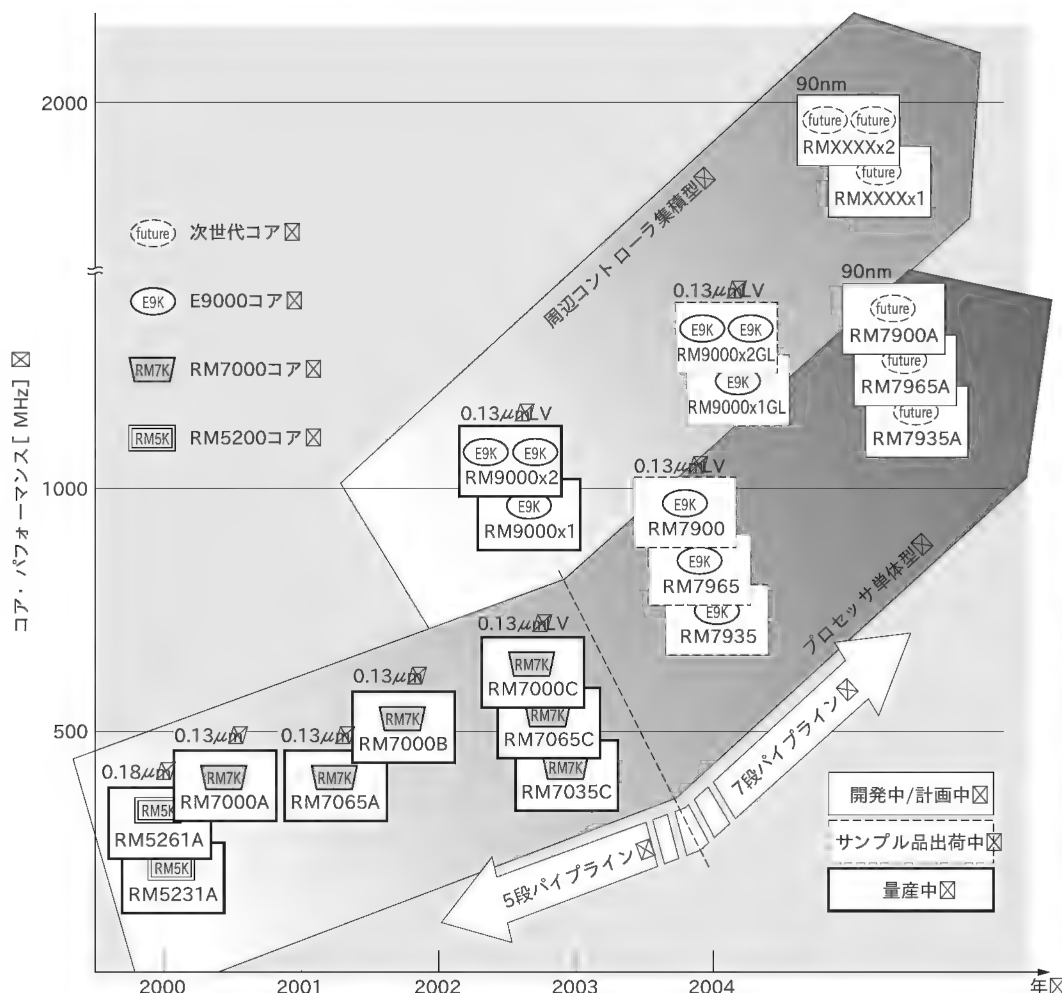


図1
PMC-Sierra MIPSベースCPUのロード・マップ

はLVTTLモードで最高133MHz，またHSTLモードでは200MHzでの動作が可能です．CPUコアは完全な64ビット・スーパ・スカラ・アーキテクチャで，二つの整数演算もしくは整数演算と浮動小数点演算を一つずつ同時に実行可能です．MIPSでは伝統的な5段パイプラインでキャッシュ・ミス時のペナルティを最小に，600MHzで1782MIPSの高いパフォーマンスを提供します．

RM7000Cは64ビットのSysADバスを搭載し，最大64Mバイトの外部3次キャッシュの制御回路も内蔵します．RM7035Cは32ビットのSysADバス，RM7065Cは64ビットと32ビットのSysADバスを選択可能です．

▶RM7900ファミリ

RM7900ファミリはRM9000ファミリ向けに開発された，独自のE9000コア(MIPS64互換デュアル発行スーパ・スカラ)を採用し，最高900MHzで2313MIPSという高いパフォーマンスを提供しながら，既存のRM5200A，RM7000Cの各ファミリからのピン互換およびソフトウェア互換を確保した新製品です．4ウェイ・セット・アソシアティブ1次キャッシュは，

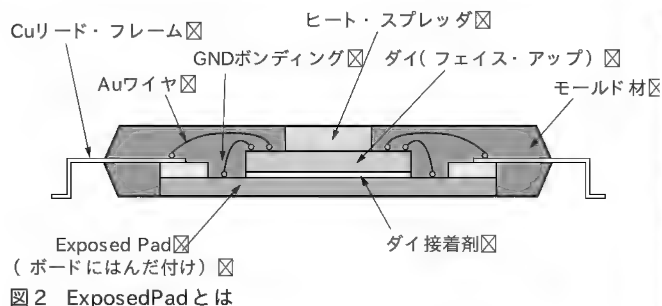


図2 ExposedPadとは

命令・データに各16Kバイトを搭載し，256Kバイトの2次キャッシュは5クロックでアクセス可能です．さらに8Kエントリの分岐予測ブロックでパフォーマンスを向上しています．システム・インターフェースはRM7000Cシリーズ同様にデバイスにより64ビットと32ビットに対応し，RM7900ではEZ Cacheプロトコル・モード対応により，タグRAMが不要な3次キャッシュ制御回路も搭載します(最大64Mバイト)．またデバッグ用にEJTAGをサポートし，分岐トレース・バッファによるリアルタイム・トレースが可能です．

表2　E9000 CPUコア主要機能

パイプライン・動作周波数	1.0GHz	
パイプライン・アーキテクチャ	デュアル発行スーパ・スカラ RISC	
パイプライン特殊機能	分岐予測ユニット	
キャッシュ階層	L1	命令　：16K バイト 4ウェイ・セット・アソシアティブ
		データ：16K バイト 4ウェイ・セット・アソシアティブ
	L2	統合　：256K バイト 4ウェイ・セット・アソシアティブ
キャッシュ特殊機能	ノン・ブロッキング・ロード プリフェッチ命令 キャッシュ・ロッキング FastPacketCache キャッシュ・コヒーレンシ	
開発サポート	EJTAGデバッグ機能 パフォーマンス・カウンタ 実行トレース・バッファ	
割り込みサポート	12×マスク可能な割り込み要求入力 1×マスク不可能な割り込み要求入力 自動割り込み優先付け，およびベクタ化	

▶RM9000ファミリ

RM9000ファミリは，PMC-Sierraでは初めての周辺コントローラ集積型のプロセッサ・ファミリです．MIPS64 ISA 互換の新開発 E9000コアを，RM9000x1では一つ，RM9000x2では二つ搭載し，コアあたり 1GHz で動作します．7段パイプラインのシンメトリックなデュアル発行スーパ・スカラ・アーキテクチャを採用し，それぞれのコアには 4 ウェイ・セット・アソシアティブの命令/データ各 16K バイトの 1次キャッシュと，5クロックでアクセス可能な 256K バイトの 2次キャッシュを搭載します．アウト・オブ・オーダ命令処理もサポートし，従来製品より 2段増えたパイプラインの周波数あたりのパフォーマンスを向上します．各 CPU コアは CPU 専用の 64G バイト /s のスイッチ・ファブリッ

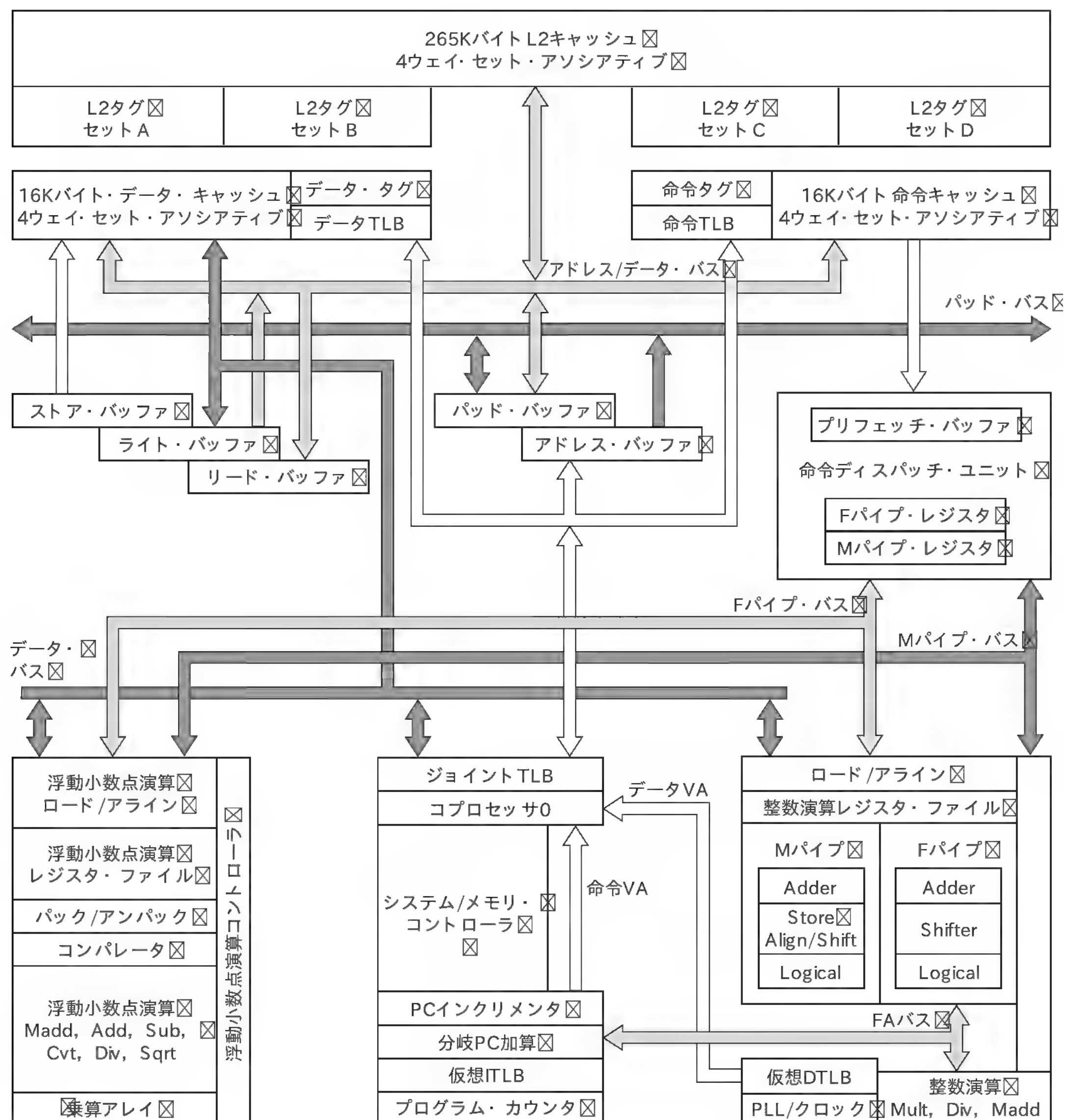


図3　E9000コアのブロック図

クに接続され，保護機能により各コアをまったく独立に動作させ，異なるOSを走らせることも可能です．

集積した周辺機能には，ECC対応200MHz DDR-SDRAMコントローラ，ローカル・バス，8KバイトのスクラッチRAM，EJTAGによるデバッグ，リアルタイム・トレースなどがあり，システム・バスは伝統的なSysADバスで過去の資産に対する互換性を確保しながら，500MHz DDR高速シリアル・インターフェース，HyperTransportによる高速な外部バスも提供します．

▶RM9000GLファミリ

RM9000ファミリに，高速通信機能を追加したものがRM9000GLファミリです．RM9000ファミリと同じE9000コアを採用し，シングル・コア（RM9000x1GL），デュアル・コア（RM9000x2GL）の製品をそろえています．RM9000ファミリの200MHz DDR-SDRAMコントローラやSysAD，HyperTransportの各インターフェースに加え，32ビットPCIも提供し汎用性を高めています．また，通信インターフェースとして10/100/1000MbpsのEthernet MACを最大3チャネル備え，通信インフラ装置で標準のPOS-PHY L3規格に似たGPI（Generic Packet Interface）としても設定可能で，POSやATM，ギガ・ビットEthernetなどの物理レイヤICやトラフィック・マネージャなどの通信ICを直接接続できます．ほかに，UARTやタイマなども集積し，ほぼワンチップでシステムが構成可能なほどの機能を提供します．

3 E9000 CPUコアの詳細

● E9000 CPUコアの概要

E9000 CPUコアは，PMC-Sierraより提供される数多くのMIPSベース・プロセッサの中でも，今後の製品を代表するコアです．表2にE9000コアの機能概要を表します．

このCPUコアは二つの製品群に搭載されています．一つはプロセッサ単体型ファミリです．この形式では，E9000コアはシステム・バス・コントローラと組み合わせて提供されています．このシステム・バスがメモリやI/Oコントローラといった外部デバイスとのすべての接続点となります．システム設計者の視点からすると，こうした形のデバイスはシステムの設計に柔軟性があり，ユーザ独自機能を高度に集積化したASICやFPGA，またMarvell社のMV64440といった汎用システム・コントローラ・チップなど標準SysADシステム・バスをもつ多くのデバイスと容易に接続できます．

もう一つの形が，E9000コアと，たとえばメモリ・コントローラ，I/Oコントローラ，DMAエンジン，オンチップSRAMなど周辺機能を同じシリコン上に集積した周辺コントローラ集積型プロセッサ・ファミリです．

● E9000コアの主要機能

E9000コアは，現在0.13μmプロセスで製造されているデバイスで，最高1GHzの動作周波数で提供されています．近い将来には90nmといった先端のプロセスで2GHzを超える周波数を実現できるようになります．

E9000コアは高いパフォーマンスを備えた組み込みシステムをターゲットとして開発されました．

主要な機能としては次のような点が挙げられます．

- デュアル発行スーパ・スカラ・パイプライン
- 2段階キャッシュ階層
- 高速浮動小数点演算
- 高性能メモリ管理ユニット

● E9000コアの構造と動作の概要

▶パイプラインとキャッシュ階層

図3にE9000コアのブロック図を示します．E9000コアの中心は，2段キャッシュ階層にサポートされた高速RISCパイプラインです．このキャッシュ階層の第1段にあるのが，4ウェイ・セット・アソシアティブで命令・データ独立のそれぞれ16Kバイトのキャッシュです．第2階層には4ウェイ・セット・アソシアティブな256Kバイトの2次キャッシュを一つ備えています．

図4に示すように，パイプラインは7段で構成されています．それぞれの主要機能は以下のとおりです．

- Iステージ：次の命令のペアをフェッチするよう命令キャッシュに要求を提示
- Cステージ：命令キャッシュ中に命令があるかチェック．必要ならキャッシュ/メモリ階層の下層から命令のリード開始
- Rステージ：可能な限り多くの命令を実行ユニットに対して発行．1パイプライン・クロック・サイクルに対し，最大2命令を発行可能

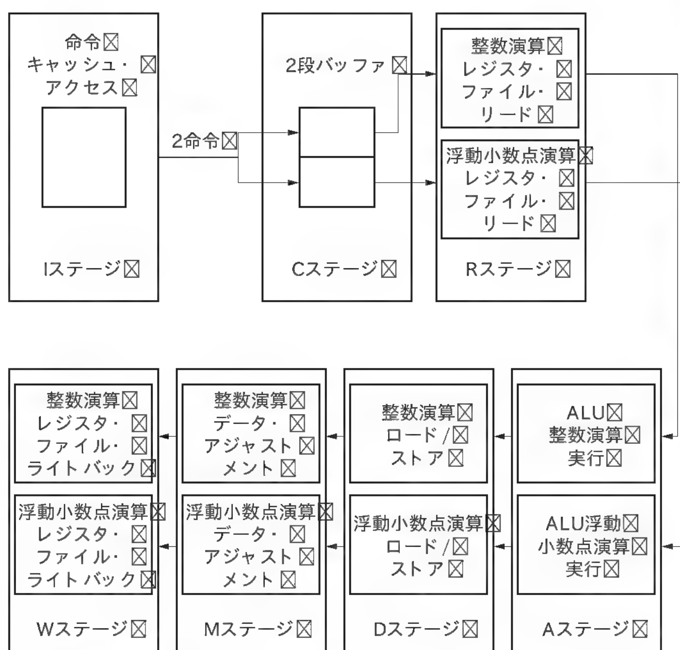


図4 E9000コアのパイプライン概要

- Aステージ：実行ユニットが命令を処理
- Dステージ：カレント命令の一つがロード/ストア要求した場合，ロード/ストア・ユニットがデータ・キャッシュにアクセス開始
- Mステージ：直近のデータ・キャッシュ要求が成功したかをチェック．必要ならキャッシュ/メモリ階層の下層からリード/ライト要求を開始
- Wステージ：次の命令ペアを回収．必要なら，その結果をターゲット・レジスタに保存

▶メモリ管理ユニット（MMU）

高速なパイプラインは高性能なメモリ管理ユニット（MMU）によってサポートされています．このユニットによりソフトウェアのタスクをシステム上の物理メモリより大きくすることができます．またソフトウェアのタスクがほかのタスクのオペレーションを保護するために，保護仮想メモリ機能を提供します．

E9000コアの仮想メモリ機能は2種類のアドレス・サイズで動作可能です．一つは仮想アドレスが32ビットで物理アドレスも32ビットの場合です．これはE9000コアが2^{32}バイト＝4Gバイトの物理メモリまでアドレス可能ということです．多くの組み込みシステムでは，これは十分なサイズです．

第2の例としては，仮想アドレスが40ビットで物理アドレスが36ビットの拡張アドレシングと呼ばれる例で，E9000コアが最大2^{40}バイト＝1Tバイトの物理メモリをアドレス可能です．これは非常に大容量のデータを処理するアプリケーション，たとえば超高精細カラー・プリンタや超高性能ネットワーク・ルータ・システムなどで有効です．

通常，アドレス変換はMMUのマイクロTLBと呼ばれる二つの小さなブロックで実行されます．そのうち一つは，命令アドレスを変換するマイクロTLBでiTLBと呼ばれています．もう一つは，データ・アドレスを変換するdTLBです．これにより，MMUが命令アドレス一つとデータ・アドレス一つの二つのアドレス変換を，同時にかつ遅延なしに実行できることになります．

このマイクロTLBには，それぞれのユニットがCPUコアのパイプラインの最大スピードで動作できるよう，非常に小さなエントリ容量しか与えられていません.この容量の小ささから，ときおりメインTLBを検索し，必要なアドレス変換情報を捜しておく必要があります．マイクロTLBがあることで，MMU自身がCPUコア・パイプラインのフル・スピードで動作する必要がなくなり，その容量も64エントリと大きくすることが可能となっています．非常にまれに新しい情報がメインTLBに導入される必要があり，その情報はオペレーティング・システムによって提供されます．

● E9000コアのパイプライン・パフォーマンス

E9000コアのパイプラインは，それぞれのキャッシュ・アクセスを2サイクルで終了可能です．命令フェッチがIステージで開始され，Cステージでタグのチェックが完了します．同様にデータの参照はDステージで開始され，タグのチェックがMステージで完了します．ここで重要なことは，E9000には2サイクルの分岐遅延と2サイクルのロード遅延があるということですが，ハードウェアおよびソフトウェアの最適化により，CPUのパフォーマンスはこれらの遅延によって下がることはありません．

分岐遅延を避けるため，分岐予測機能が盛り込まれています．このユニットはターゲットのアドレスがすべてのジャンプと分岐命令に関してモニタし，そのパターン情報（分岐したか否か，次の命令フェッチのアドレス）を保存しています．分岐予測ユニットはこの情報を使い，分岐もしくはジャンプに続く次の命令フェッチのアドレスをほぼ正確に予測し，2サイクルの分岐遅延を回避しています（図5）．

ロード遅延はE9000コアのノン・ブロッキング・ロード機能で回避可能です．図6に示すように，ロード命令が命令キャッシュから読み出す値をターゲット・レジスタに置くまでに2サイクルかかります．

この2サイクル間に命令がそのソースのオペランドとしてターゲット・レジスタを読み出す必要がないという条件を満たす限り，CPUはほかの命令を処理できます．コンパイラが命令を最適な順序に並べ替えることで，この機能を最大限に活用できます．ノン・ブロッキング・ロード機能の詳細については後述します．

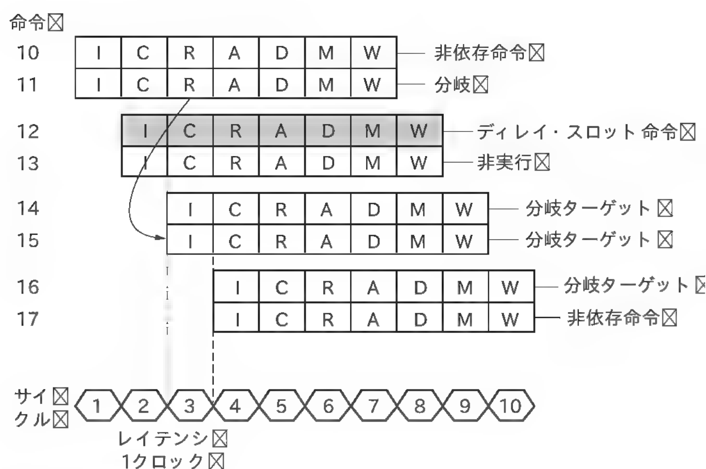


図5　分岐遅延のメカニズム

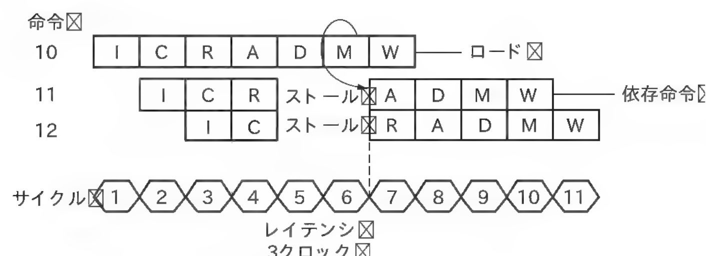


図6　ロード遅延のメカニズム

● MMUの独自機能 ―― 巨大なTLBページ・サイズ

多くのMIPS CPUでは，TLBエントリ一つでマッピング可能な最大のページ・サイズは16Mバイトです．

E9000コアは64Mバイトを超え，さらに256Mバイトまでの大きなページ・サイズが使用可能です．これは多くの組み込みシステムにおいて有効で，システムの初期化中にTLBエントリのセットアップを完了することができ，通常の動作中には変更する必要がなくなります．

● E9000キャッシュ階層の独自機能

▶ ノン・ブロッキング・ロード

ノン・ブロッキング・ロードのサポートがないCPUでは，キャッシュ・ミスが起きた場合，どのような命令でもパイプラインをストールさせてしまい，システムのパフォーマンスを著しく低下させます．E9000コアはノン・ブロッキング・ロードでこの状況を高確率で回避できます．

キャッシュ・ミスを引き起こすロード命令を実行する際には，E9000のパイプライン・ロジックは直接2次キャッシュに，また必要であればシステムのメイン・メモリに要求をパスし，この処理中にCPUパイプラインは遅延なく動作できます．E9000のキャッシュ制御ロジックはCPUコアのこうした動作を許容しており，キャッシュやメモリで二つのロード命令を処理中でもCPUコアの正常動作も可能です．この場合，キャッシュ階層からロード要求の結果が戻ってくる前にCPUがその結果を使おうとしないことが正常動作の条件です．使いたい場合，やはりパイプラインを待つ必要がありますが，コンパイラが命令を並べ替えて影響が最小になるように調整してくれます．

▶ プリフェッチ命令

CPUがプリフェッチ(PREFETCH)命令を実行すると，必要な情報をデータ・キャッシュにロードする要求を開始します．2次キャッシュもしくはメイン・メモリ・コントローラが要求されたデータを準備する間も，CPUパイプラインは遅延なしに動作し続けます．

この命令は，データ・キャッシュにはないデータを扱う際に有効です．実際のソフトウェアでは，データを実際に処理するプログラムの部分にさしかかる少し前のコードにプリフェッチ命令を入れることになります．これでプリフェッチ命令がデータ・キャッシュにデータを事前に格納可能で，CPUがデータを処理する命令を実行するときには，必要な情報がすべてデータ・キャッシュにそろっており，遅延なしに処理が進行可能になります．

▶ キャッシュ・ロッキング機能

この機能を使用することで，コードやデータのもっとも重要な部分をキャッシュ・ラインに閉じ込め，ほかの動作により置換されないよう保証できます．これでCPUコアがこのロックされた情報を参照する限り，かならずキャッシュがヒットし，短いアクセス・タイムが保証されます．

▶ ファースト・パケット・キャッシュ機構

多くのシステムで，ある種の情報は2次キャッシュにできる限り長く留め，ほかの動作による不用意なデータ置換がないようにすることは有効です．たとえばネットワーキング・システムでは，ソフトウェアがフォワーディング・テーブルという大きなデータ構造体を頻繁に参照します．この構造体は新しく受信したネットワーク・パケットをどのような経路に渡すべきかという重要な情報を保存しています．

通常CPUが効率よく必要な情報にアクセスできるよう，フォワーディング・テーブルの情報は2次キャッシュ内に格納することが望ましいのですが，CPUが新しいネットワーク・パケットを受信し，データ・キャッシュにロードすると，通常はこの動作で2次キャッシュ中のフォワーディング・テーブルの一部を置換してしまい，システム効率の低下につながります．

ファースト・パケット・キャッシュにより，E9000コアは，2次キャッシュに影響与えることなくネットワーク・パケット情報をデータ・キャッシュに読み込めます．よってフォワーディング・テーブル情報が保持されて，システム効率がつねに最高レベルに保たれるのです．

▶ キャッシュ・コヒーレンシ

E9000コアはキャッシュ・コヒーレンシに対して独自のサポートを提供しています．2次キャッシュ・コントローラがタグ・メモリを2セットもっており，1セットは通常の方法で使用され，もう1セットはコヒーレンシのサポートに使われるシャドウ・タグと呼ばれます．組み込みシステムでは，外部デバイスからシステム・メモリに新しい情報を移動するために，DMAコントローラを使用するのが一般的ですが，時折DMA転送により影響を受けたメモリ領域に，以前あった情報のコピーがキャッシュに残っていることがあります．

DMAコントローラが転送を完了した後は，新しい情報がシステム・メモリに存在します．しかしCPUにコヒーレンシのサポートがない場合には，すでに無効になってしまったにも関わらず以前の古いメモリの内容がキャッシュに保存されている可能性があります．

CPUの情報が古い場合には，ソフトウェアがキャッシュ・ラインを確実に更新し，古い情報に「無効」とマーキングしなければなりません．もしこの動作が正確になされなければ，システム動作の後ほどにCPUが古い情報を使おうとし，理解するまでに非常に苦労するエラーを発生させる原因となります．

E9000コアのキャッシュ・コヒーレンシのサポートにより，こうした問題を容易に回避可能です．DMAコントローラがシステム・メモリに新しい情報を書き込むときに，E9000のキャッシュ制御ロジックがシャドウ・タグを使ってキャッシュに古いバージョンの情報がないかをチェックします．もし，あった場合には，自動的にキャッシュ・ラインに無効マーキングし，後に古い情報を使ってエラーを起こすことを防ぎます．

さらに，E9000コアはダイレクト・デポジットと呼ぶ独自機

能によって，より効率的にDMA転送を管理します．この機能によりDMAコントローラがメイン・メモリに転送した情報は自動的に2次キャッシュにコピーされ，CPUはメイン・メモリから読むよりも小さい遅延で情報の参照が可能となります．

この機能には二つのモードがあります．一つはオート・デポジットというモードで，外部のDMAコントローラが情報のブロックをメイン・メモリに書き込んだ際に，この機能が自動的にブロックの最初の部分を2次キャッシュにコピーします．ここでコピーされるバイト数は任意に設定が可能です．

このモードはネットワーク・インターフェースからトラフィックを受信するシステムで有効です．オート・デポジット機能を使えば，CPUが小さな遅延でパケットのヘッダを参照でき，ネットワーク処理が効率的に実行可能です．

もう一つのモードはライブ・デポジットといい，外部DMAコントローラがシステム・メモリに書き込むバイトが2次キャッシュにコピーする必要があるかどうかを自動的に判断してくれます．これはデバイス間でメッセージを受け渡すシステムで有効で，CPUに効率良くメッセージを読み込ませることが可能になります．

● E9000コアのそのほかの機能

▶CPUコアのコンフィグレーション

E9000の動作には，たとえばシステム電源を投入したときのリセット後，CPU動作開始時に設定できる要素がいくつかあります．具体的には，E9000コアは多くのネットワーキング・システムで便利なビッグ・エンディアンでも，多くのPCIベースのシステムで便利なリトル・エンディアンでも動作が可能です．また，E9000コアを含む多くのデバイスは入力されるクロック信号を逓倍してパイプライン・クロックの周波数を生成しています．逓倍数はコンフィグレーションの一部として設定可能で，システム設計者は必要以上の電力を消費することなく必要なパフォーマンスを提供する動作周波数を選択できます．

CPUはこうしたコンフィグレーション情報をブート時にシリアル・ストリームから読み込み，通常動作を始める前に設定します．

▶ システム・デバッグ・サポート

E9000コアはMIPSプロセッサ標準のEJTAGでのデバッグをサポートしています．JTAGインターフェースを通してE9000ベースのシステムにコードをダウンロードすることができ，プログラムのソース・コードを参照しながらブレーク・ポイントを設定するなど，ソフトウェア・ツールの機能を駆使することで，ソフトウェアのバグの発見を支援します．

E9000コアはブレーク・ポイントの設定方法が二つあります．一つはEJTAG規格にある`SDBBP`命令です．この命令で例外ハンドラや割り込みサービス・ルーチンなど，多くのデバッグ・ツールが対応していないソフトウェアの部分も分析可能になります．このサポートがないMIPSベースCPUでは，ブレーク・ポイントは`BREAK`命令によって設定しなければなりません．難しい点は，この命令がハードウェア割り込みなど例外処理で使われているCPUレジスタの値を変えてしまうため，`BREAK`命令による例外ハンドラ・ソフトウェアのデバッグが非常に難しいということです．`SDBBP`命令は例外ハンドリング・レジスタの値を変化させないので非常に簡単です．

E9000でブレーク・ポイントを設定するもう一つの方法は，ウォッチ・ポイント・レジスタの一つを使用することです．ブレーク・ポイントの仮想アドレスがウォッチ・ポイント・レジスタに書き込まれます．また，ウォッチ・ポイント・レジスタのほかのいくつかのビットはそのブレーク・ポイントがCPUの命令フェッチのためなのか，データの参照のためなのか，インデックスを設定可能です．ウォッチ・ポイント機能がイネーブルされた後，CPUが示すすべてのアドレスがウォッチ・ポイント・レジスタの値と比較対照されます．マッチした場合にはそのブレーク・ポイントはアクティブになり，システム内部の状況が解析できるようにソフトウェア開発ツールに制御権が返還されます．

▶ パフォーマンス解析サポート

E9000コアは二つのパフォーマンス・カウンタを内蔵し，ソフトウェアによって読み込むことが可能です．それぞれ任意の形式のイベントをカウントするよう設定できます．

たとえば，一つのカウンタは全実行命令数をカウントするよう設定し，もう一つは命令キャッシュのミスをカウントと設定し，システム・ソフトウェアの一部を動作後二つのカウンタ値を読み出し命令キャッシュのヒット率を計算できます．これはソフトウェアのパフォーマンス改善点を探るための便利な使いかたで，システム全体のパフォーマンスの向上に役立ちます．

▶ 実行トレース・オプション

E9000コアを使ったデバイスにより，CPUの命令実行パターンに関する情報を集める小さなストレージ・バッファをもったものがあります．たとえば，このバッファはCPUコアによって実行された命令のアドレスをストアするよう設定できます．これは，設計者がCPUコアのバス信号に直接ロジアナを接続できない集積型デバイスの挙動を理解するために便利です．

4 プロセッサ単体型CPU-RM7900ファミリの詳細

● RM7900ファミリの概要

RM7900ファミリの概要を表3に示します．このファミリはE9000コアをPMC-SierraのRM7000ファミリと互換のパッケージに封止したデバイス・ファミリです．このパッケージの互換性により，RM7000ファミリ向けに設計したシステムは，容易にRM7900ファミリの高いパフォーマンスを享受できます．

RM7900ファミリは，RM7000ファミリにはなかったEJTAG開

発/デバッグ・サポート機能を提供しているので，この方法でシステムを容易にアップグレードすることができます．

RM7900ファミリ・ベースの一般的なシステムの例を図7に示します．このデバイスは浮動小数点ユニット，MMU，2段階キャッシュ階層を内蔵したE9000コアとSysADバス・コントローラ，コンフィグレーション用シリアル・インターフェース，およびEJTAG回路が集積されています．例として掲載したシステムでは，RM7900ファミリのSysADバスはシステム・コントローラに接続され，さらにはSDRAM，フラッシュ・メモリ，ネットワーク・インターフェース，プリント・エンジンなどのそのほかの周辺装置と接続しています．

● RM7900ファミリの特殊機能

▶SysADバスに高速なHSTLモードを追加

RM7900ファミリのシステム・バスはSysADと呼ばれ，64ビットもしくは32ビットのデータ幅でデータとアドレスが多重されたインターフェースです．通常のトランザクションでは，RM7900はアドレスと制御信号を発信しブロック・リード要求やシングル・ライト要求といった要求の型を表示開始し，次のサイクルでデータ・バスがリード・データもしくはライト・データを転送します．このSysADはMIPSでは標準的です．

RM7900ファミリのSysADインターフェースは，二つのモードで動作可能です．一つは，LVTTL信号レベルを使用し，一般的なコンパニオン・チップで使用されているレベルと互換性があります．このモードではSysADインターフェースは最大133MHzで動作可能です．二つ目のモードでは，HSTLの信号レベルを使用し，小さな電圧振幅により動作周波数を最大200MHzまで上げられます．この高速動作では，64ビットのSysADインターフェースは1600Mバイト/sもの広帯域となり，システム・パフォーマンスを大幅に向上させることができます．

▶EZ-Cacheモード

高速に非常に大容量のデータ処理が必要なシステムの場合には，外部に3次キャッシュを搭載することが可能です．もちろんコストがかかるので，使うかどうかはシステム次第です．RM7900は外部3次キャッシュ制御ロジックをすべて内蔵しているので，SRAMデバイスの接続だけでキャッシュが実現できます．

RM7900の3次キャッシュ・コントローラは，二つのモードがあります．一つは，3次キャッシュのタグ値をタグRAMと呼ばれる特殊外部メモリに保存して，これをチェックして使うモードです．これはRM7000とまったく同じ動作なので，RM7000を使用したシステムをそのままアップグレードするのに便利です．

もう一つのEZ-Cacheモードは，RM7900の3次キャッシュ・コントローラ内部にタグ値を保存し比較できるので，外部に特殊なタグRAMを置く必要はなくなり，標準のSRAMのみで3次キャッシュが実現できます．

3次キャッシュは外部のSRAMで実現するため，内部の1次キャッシュや2次キャッシュに比較して高速に動作できませんが，最大64Mバイトの容量を搭載可能という利点があります．非常に大容量のデータを処理するシステムでは，CPUが必要とする情報はほとんど3次までのキャッシュ階層で見つけられることになります．

3次キャッシュはメイン・メモリのSDRAMと比べると格段に高速なアクセスができるのでパフォーマンス面で大きな利点となります．

▶RM7000ファミリとのソフトウェア互換性

ソフトウェア・エンジニアの視点からすると，小さな変更でRM70xxCからRM79xxにシステムをアップグレードすることが可能です．CPUの制御レジスタなどの重要な機能のほとんどが同じで，変更が必要ありません．通常，変更が必要となるのはTLBのアップデートを制御するOSの機能など，特殊なコードの一部のみです．一般的には，RM70xxのソフトウェアでOSの一部ではない部分は，RM79xxでそのまま動作可能です．しかし，得られる限りの最高のパフォーマンスを求める場合には，E9000のパイプラインに対してコードを最適化できるツール・チェインで再コンパイルすることをお勧めします．PMC-Sierraは必要なサポートを含んだGCCのバージョンを作成しています．

表3　RM7900 CPUファミリの概要

型式		RM7900, RM7965, RM7935
CPUコア		E9000 CPUコア
パイプライン動作周波数		最大900 MHz（668, 750, 835, 900）
SysADインターフェース	データ幅	RM7900：64ビット RM7965：64/32ビット RM7935：32ビット
	クロック	200 MHz（HSTL） 133 MHz（LVTTL）
外部3次キャッシュ・サポート（RM7900のみ）		最大64Mバイト （RM7000互換EZ-Cache Modeサポート）
パッケージ		RM7900　304TBGA RM7965　256TBGA，216 ExposedPad RM7935　128 ExposedPad

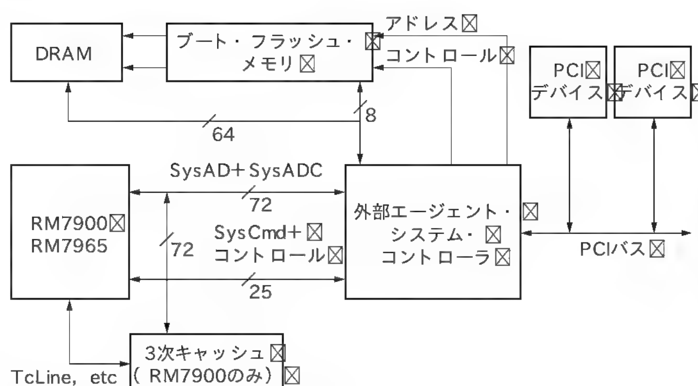


図7　RM7900ファミリ・システム例

5 周辺コントローラ集積型CPU-RM9000x2GLの詳細

RM9000x2GLはE9000コアを複数と数多くの高性能周辺機能をワンチップに集積し，多くの利点を提供します．高い集積度により機能ユニット間の超高速通信が可能となり，また組み込みシステム向けに消費電力も低減できます．RM9000x2GLはCPUコアを二つ，RM9000x1GLはCPUコアを一つ集積し，周辺機能によりそれぞれ3種類ずつ製品があります．RM9000x2GLのおもな機能を表4に示します．

表4 RM9000x2GL CPUファミリ　主要機能

型式	RM9220, RM9222, RM9224（デュアル・コア）
	RM9120, RM9122, RM9124（シングル・コア）
CPUコア	E9000 CPUコア（それぞれに2次キャッシュ搭載）
パイプライン・クロック周波数	最大1GHz（800M, 1000M）
インターフェース・コントローラ	200MHz DDR-SDRAMコントローラ 最大3チャネル× 10/100/1000 Base Ethernet MAC 200MHz SyaADインターフェース 16 Gbps HyperTransport（HT）コントローラ 32ビット PCIコントローラ（33 / 66MHz） 16MHz ローカル・バス・コントローラ
内部接続	5ポート・パケット・スイッチ・ユニット ●全ポートに対してコンカレントな転送をサポート ●他の内部ユニットに対しデータを転送
他の内部機能	中央割り込みコントローラ 4チャネル 汎用DMAコントローラ 8Kバイト・スクラッチRAM
パッケージ	RM9220, RM9120, RM9222, RM9122　672 FCBGA
	RM9224, RM9124　896 FCBGA

● RM9000x2GLの特徴

RM9000x2GLデバイスは，二つのE9000コアで構成されるCPUサブシステムと高いパフォーマンスのDDR SDRAMコントローラ，各種高速インターフェース・コントローラで構成されています．これらのブロックはパケット・スイッチと呼ばれる中央の相互接続ユニットで内部接続されています（図8）．

すべての機能ユニットは，RM9000x2GLの物理アドレス空間の領域として見えます．各機能ユニットのアドレス領域は，システムのブート・ストラップのシーケンス中にデバイス内部のレジスタに値を書くことで，ソフトウェア制御で選択します．RM9000x2GLの物理アドレス空間は，内部CPUのどちらからも，またネットワーク・インターフェースなどほかの内蔵機能ユニットからもリード/ライト可能です．さらにHyper Transportなどのインターフェースに接続された外部デバイスからもリード/ライト可能です．

RM9000x2GLはコンカレンシの考えかたを取り入れ，一つずつシステム・アクティビティをこなしていくのではなく，数多くのアクションを同時に実行できます．これにより遅延を排除し全体のシステム・パフォーマンスの向上を実現しています．この考えかたはそれぞれの機能ユニットやそれらの相互通信を補助するロジックの設計に取り入れられています．

● デュアルCPUサブシステム

RM9000x2GLは，CPUサブシステムに完全なE9000コアを

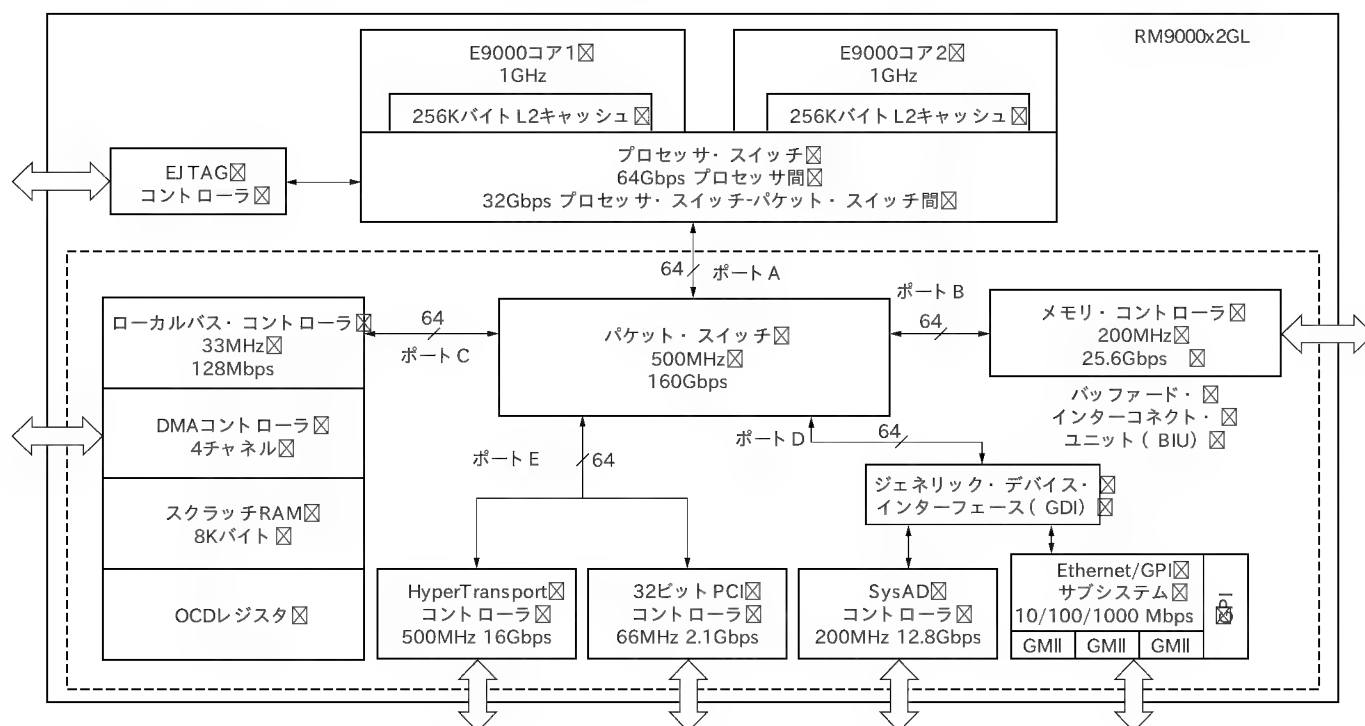


図8 RM9000x2GLのブロック図

二つ備えています．それぞれのCPUコアには1次命令/データ・キャッシュおよび2次キャッシュを搭載しています．

各コアの動作は完全に独立しているので，それぞれ別々の動作が可能です．非常に演算処理要求の高いシステムでは，二つのコアは協同し合うことで演算を分け合い効率的な計算が可能です．逆に，まったく違ういくつかのタスクがからむようなシステムでは，いくつかはこちらのCPUに，残りはこちらに，とタスクを割りふる方法が効果的です．どちらのケースでも，RM9000x2GLは優秀な演算能力を発揮します．

二つのE9000コアは専用高速バスで相互接続され，二つの主要な機能があります．まず，CPUと中央のパケット・スイッチ間の接続をします．コアの一つがSDRAMにアクセスする場合，もしくは情報を周辺機器インターフェースの一つから転送する場合，このバス経由でパケット・スイッチに要求を提示します．

二つ目には，キャッシュ・コヒーレンシのトランザクションに必要なデータ・パスとして機能します．前述のとおり，I/O転送でキャッシュ中の情報が陳腐化すると，I/Oコントローラが自動的にキャッシュ内の情報を無効化することが可能です．I/O転送の詳細はパケット・スイッチからこのバス経由でCPUコアに伝達されます．CPUコア中のキャッシュ・コントローラがこの情報を利用しキャッシュ・ラインをチェックし，必要なら無効化します．

● DDR-SDRAMコントローラ

RM9000x2GLのDDR-SDRAMコントローラは，CPUおよびI/Oコントローラが使用できる高速なメイン・メモリ・サブシステムを構成しています．DDR-SDRAMコントローラは最大200MHzで動作します．サイズは64Mビットから1Gビット，またデータ幅は8ビットから32ビットまでの各種SDRAMデバイスをサポートします．SDRAM自体はバッファ・タイプでもレジスタ・タイプでも対応し，ECC機能もサポートします．

非常に高速なメモリが必要なシステムでは，非常に速いクロックで動作し，低いCAS遅延のSDRAMを使う必要がありますが，高速デバイスのコストは低価格のシステムにとって大きな負担となります．RM9000x2GLは内部のレジスタの設定によりCAS遅延などタイミング・パラメータを設定可能なので，低速だが安価，高価だが高速など，システムに合わせてSDRAMの選択が可能です．

32ビット・データ幅の64MビットSDRAMが二つだけで構成されるシングル・バンクで合計16Mバイトと，容量は少ないが広帯域という選択もでき，逆に超大容量のメモリが必要なシステムでは，8ビットのデータ幅で1GビットのSDRAMデバイス8個構成のメモリ・バンクを4バンクまで使えます．デバイスあたり128Mバイトなので，1バンクで1Gバイト，4枚で4Gバイトの大容量をサポートします．

RM9000x2GLの洗練されたSDRAMコントローラでは，まずメモリに入る要求はキューに保存され，複数の要求の関係性を注意深く比較し，最適な順序に実行されます．たとえば，メモリ・コントローラがリード要求をCPUコアの一つから受け取り，直後にギガ・ビットEthernetポート専用DMAコントローラからライト要求を受け取った場合，シンプルなコントローラではCPUが発したリード要求が必要な動作をすべて完了する間，ギガ・ビットEthernetからのライト要求は強制的にウェイトが入ります．

RM9000x2GLではメモリ・コントローラが，二つの要求が違うメモリ・バンクに関わるかを確認し，バンクが違う場合にはライト要求を待たせる必要はありません．一つのバンクでリード処理中でも同時に別バンクでライトが開始されるのは問題ありません．この方法でRM9000x2GLは遅延の多くを回避可能で，システムのパフォーマンス向上に大きく貢献します．

● I/Oインターフェース・コントローラ

型式により最大三つのEthernetインターフェースを搭載し，専用DMAコントローラはEthernetインターフェース間やメイン・メモリとの転送を自動管理します．このインターフェースは10M/100M/1000Mbpsに対応し，MII，GMII，TBIの各インターフェースでPHYデバイスに接続します．また，通信装置で標準のPOS-PHY規格ベースのGPI(Generic Packet Interface)に設定でき，POS-PHY L3準拠のトラフィック・マネージャやPHYなどを直結できます．前述のオート・デポジット機能でEthernet MACで受信したパケットのヘッダを専用DMAエンジンが自動的に2次キャッシュに，またペイロードをメイン・メモリに格納するなど効率的なキュー管理とチェック・サム計算，パケット・ヘッダの加工をサポートします．

RM9124，RM9224，RM9120，RM9220でサポートするHyperTransportインターフェースは，ソース同期のポイント-ポイントのパケット・ベース高速シリアル・インターフェースです．HyperTransport規格ではデータ幅を2ビットから32ビットまで規定していますが，RM9000x2GLでは8ビットの片方向チャネルを送受信で一対もっています．500MHzのDDRで動作する差動信号を使用し，データに8ビット，クロックと制御ビットをあわせて片方向20ピンと少ないピン数のインターフェースです．

これによりHyperTransportインターフェースは，最大双方向で16Gbpsという非常に広帯域なI/O転送を実現し，プリンタやネットワーク装置など大容量のデータ転送が必要なアプリケーションに対してつねに有効です．

RM9124，RM9224，RM9122，RM9222の各デバイスは，32ビット/66MHzのPCI 2.1対応コントローラを内蔵します．二つの内蔵CPUに対してはホスト・ブリッジ機能を提供し，内部DMAコントローラおよびSysADに接続する外部デバイスに対してはマスタとして機能します．またDDR-SDRAMコントローラやスクラッチRAM，SysADの空間やOCDレジスタに対しては，ターゲットとして四つの遅延トランザクション・

バッファを提供します．マスタ・デバイスを五つまで調停可能で，うち一つは内部のマスタを管理します．5Vトレラントな3.3V I/OでPCI-PCIブリッジとしての基本機能も提供し，PCI Haltモードのサポートにより活線挿抜も可能で，不活性トランザクションの検出もサポートします．

SysADインターフェースはRM7000と使うために，過去に開発された64ビット・システム・コントローラ・デバイスと容易に接続することが可能です．Ethernet機能を使用しない場合に使えるバスで，HSTLで最高200MHzをサポートします．

こうしたすべてのインターフェース・コントローラは，CPUコアによる最低限のサポートでSDRAMメモリとデータをやりとりできます．それにより，I/Oコントローラがシステムのデータの動きを管理している間に，CPUが影響を受けずに高度な計算処理をそのまま続けられるのです．

● パケット・スイッチ

パケット・スイッチはシステムのすべての機能ユニット間通信を制御し，RM9000x2GLの非常に重要な部分に位置付けられています．今まではCPU，メモリ，I/Oコントローラ間のトラフィックにはシェアード・バスが多く使われてきましたが，最近では転送されるデータ容量もシェアード・バスでの設計ではサポートしきれないほど大きくなってきています．

RM9000x2GLでは，パケット・スイッチによりこの問題を解決しています．パケット・スイッチは前の転送が進行中でも，制御ロジックが待機中の転送が開始できる条件を数多く認識でき，たとえばCPUコアがSDRAMをリード中に，HyperTransportに接続された外部デバイスが内部のスクラッチRAMにライトすることも可能です．

複数のデータ転送をコンカレントに行うよう転送機会を並べ換えることのほかに，パケット・スイッチのロジックは最大400MHzの動作でシステム全体のパフォーマンスを引き上げています．同等の機能をASICやFPGAで実現するのはなかなか困難ですが，RM9000x2GLを使えば高性能な製品を容易に早く市場に出すことができます．

● そのほかの内蔵機能

RM9000x2GLは中央割り込みコントローラ(CIC)を内蔵し，デバイスの割り込みアサインを柔軟に設定できます．RM9000x2GLは割り込みを，デバイスの10個の割り込み要求入力ピンのうちどれかに直接入力される信号，またはHyper Transportインターフェースを経由して伝送された256個の割り込み要求メッセージという形で認識します．これらの割り込み要求メッセージはHyperTransportインターフェースに接続された外部デバイスもしくはRM9000x2GL自身のCPUの一つが生成し，もう片方のCPUに対する割り込み要求として伝送されます．CICの設定により，割り込みは一つ一つE9000コアのどちらか，もしくは両方をターゲットにできます．さらに各CPUコアにある12のハードウェア割り込み要求入力をどちらかのコアに個別にアサインできます．たとえば一つのCPUコアを割り込みの扱い専用に，もう一方を計算タスク専用にアサインすることもできます．逆に二つのコアに割り込みの扱いをシェアさせるように設定することもできます．

スクラッチRAMは，システム・ソフトウェアが使用できるように用意した汎用の小さな高速メモリです．システムがネットワーク・パケットを送受信する際に，スクラッチRAMはパケット・ヘッダを一時保管するのに非常に便利な場所です．

汎用DMAコントローラは4チャネル・サポートされており，四つのDMA転送が同時に実行可能です．調停ロジックを内蔵しているので，一つのチャネルでのDMA転送がほかのチャネルの転送もしくはほかの機能の動作をブロックしないよう調整します．各チャネルはCPUコアの一つが開始アドレス，宛て先アドレス，転送長をプログラムします．そしてCPUが転送開始のコマンドを発行します．この後DMAチャネルがCPUを介さずに自分で残りの手順を管理します．転送終了と同時に，転送終了を示すために，チャネルが一つもしくは両方のCPUコアに対し割り込みを発生します．

また，チェイニングと呼ばれる機能もサポートし，CPUコアが複数のDMA転送情報を逐一コントロールする必要がなくなります．このモードでは，CPUコアがメモリに実行する転送のリストを作成し，DMAコントローラにその開始位置を指示するだけで，DMAコントローラが転送の情報をメモリのリストから自動的に読み込んで，全転送が終了するまで実行を続けます．このモードでは，CPUみずから干渉することなくDMAコントローラに長く複雑な転送シーケンスを実行させることができます．たとえば，プリンタでは，メイン・メモリからレーザ・プリンタ・エンジンへの複雑なデータ転送をDMAコントローラに全面的に委託することで，CPUはプリンタ・エンジンに送る次のページを準備することに専念できるわけです．

まとめ

PMC-SierraのE9000 CPUコアは，非常に高いパフォーマンスの組み込みシステムを実現するために多くの機能を提供します．PMC-Sierraは，この高性能なコアをスタンド・アロンのCPUとしても高度に集積化されたデバイスとしても提供し，多くの組み込みシステムで応用できる製品です．

高性能なPCやディジタル機器が市場に行き渡り，ユーザはすべての製品にPCのような操作性と高性能を要求するようになり，制約の多い組み込み機器開発者はジレンマに陥っています．PMC-Sierraの高性能CPUはこうしたジレンマを解決するソリューションを提供します．これからの開発にぜひ一度，検討してみてください．

ポール・コッブ　PMC-Sierra，Inc.
ながしま・もとき　(株)アルティマ

第2部 Chapter 4　Au1 CPUコアの特徴から周辺機能，初期化手順の詳細まで

Alchemyソリューション SoCの詳細

伊藤 信

AMDは，Opteron，Athlon，Duronといった PC向けのプロセッサ，およびSpansionブランドのフラッシュ・メモリを出荷している．だが，組み込み専用のプロセッサも手がけていることは意外と知られていない．AMDの第3の事業部，PCSG（Personal Connectivity Solutions Group）は，組み込み向けのプロセッサ製品を製造・出荷している．AMD/PCSGは，2002年にAlchemy Semiconductor社を，2003年にナショナルセミコンダクターからGeode製品を事業買収し，x86およびMIPSアーキテクチャの組み込み用途向けプロセッサを提供している．本章では，AlchemyのMIPSベースSoCについて解説する．（筆者）

はじめに

現在，AlchemyのSoCとして，Au1000，Au1100，Au1500，Au1550の4種類が出荷されています（図1）．

なお，以降で，四つの製品に共通する部分に関しては，Au1xxxと記述します．余談ですが，Alchemyとは錬金術のことで，それによってできあがった製品ということでAu（金の元素記号）が型番の頭に付いています．

図1からわかるとおり，すべての製品はAu1000が元になっています．最初に開発されたAu1000から，大きく二つの方向で後継製品が開発されています．

基本になるAu1000のブロック図を図2に，Au1000，Au1100，Au1500の内蔵ペリフェラルの違いを図3に示します．

Au1500は，Au1000にPCIバス・コントローラを搭載し，拡張性を高めたものです．チップの面積を変えずにPCIを載せるために，いくつかの内蔵ペリフェラルを外しています．Au1000とAu1500はEthernetのMACを二つ搭載しており，ネットワーク機器に多く採用されています．

Au1500の後継として開発されたAu1550は，IPSec用のセキュリティ・エンジンを搭載し，ネットワーク機器や，暗号化が必須なアプリケーションに向いています．0.13μmプロセスにより，PCIやセキュリティ・エンジンを搭載しながら，Au1000と同等の消費電力を実現しています（図4）．

モバイル製品向けに開発されたAu1100は，Au1000からEthernetのMACとUARTを一つずつ外し，LCDコントローラとSDカード・インターフェースを搭載しています．また，

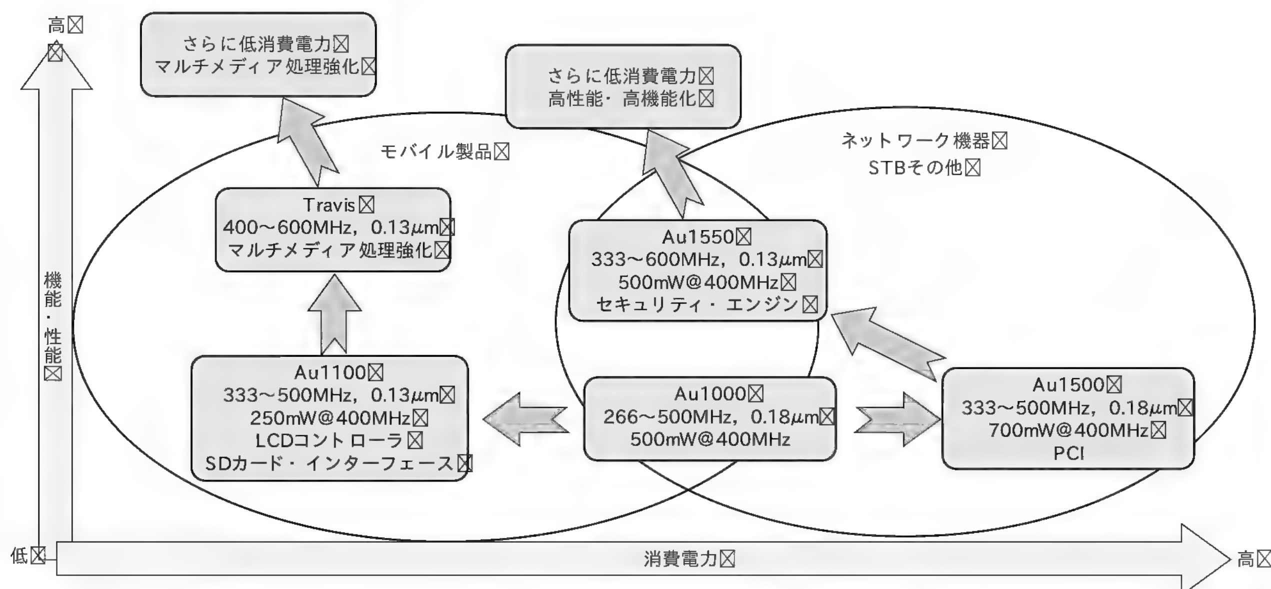


図1　Alchemyシリーズ製品のロード・マップ

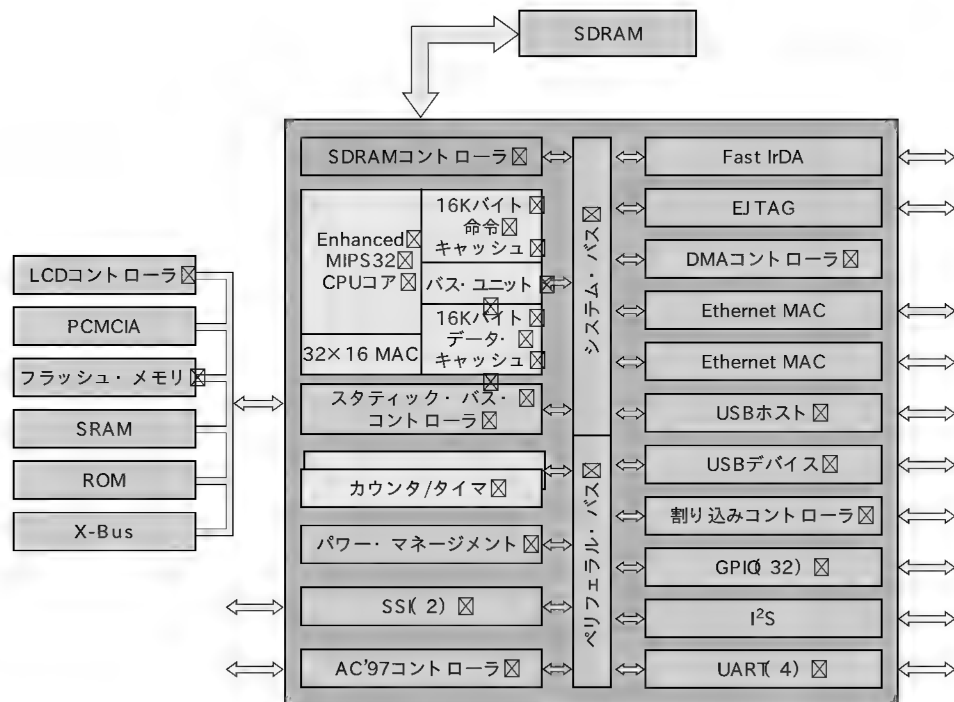


図2　Au1000のブロック図

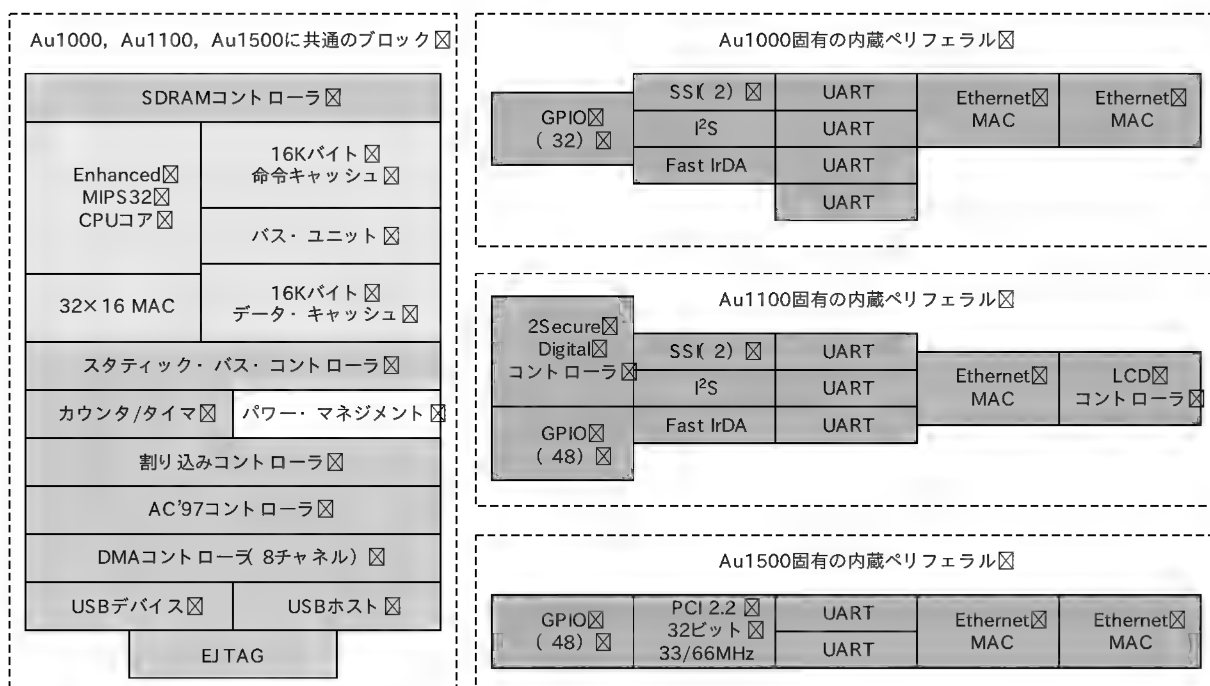


図3　Au1000，Au1100，Au1500の違い

0.13μmプロセスにより，消費電力はAu1000の半分になっています．これにより，PDAなどのモバイル製品向けのSoCとしては，高い性能対消費電力比を実現しています．Au1100の後継デバイスであるTravisは，消費電力を抑えたたまま，マルチメディア処理機能を強化した製品です．

本章では，最新デバイスであるAu1550を中心に，Alchemyソリューション SoCの特徴を紹介します．

1　Alchemy Au1 CPUコアの特徴

Alchemyソリューション SoCは，図2に示したAu1000のブロック図に代表されるように，Au1 CPUコア，システム・バス(SBUS)，SDRAMコントローラ，スタティック・バス・コントローラ，およびさまざまな内蔵ペリフェラルによって構成

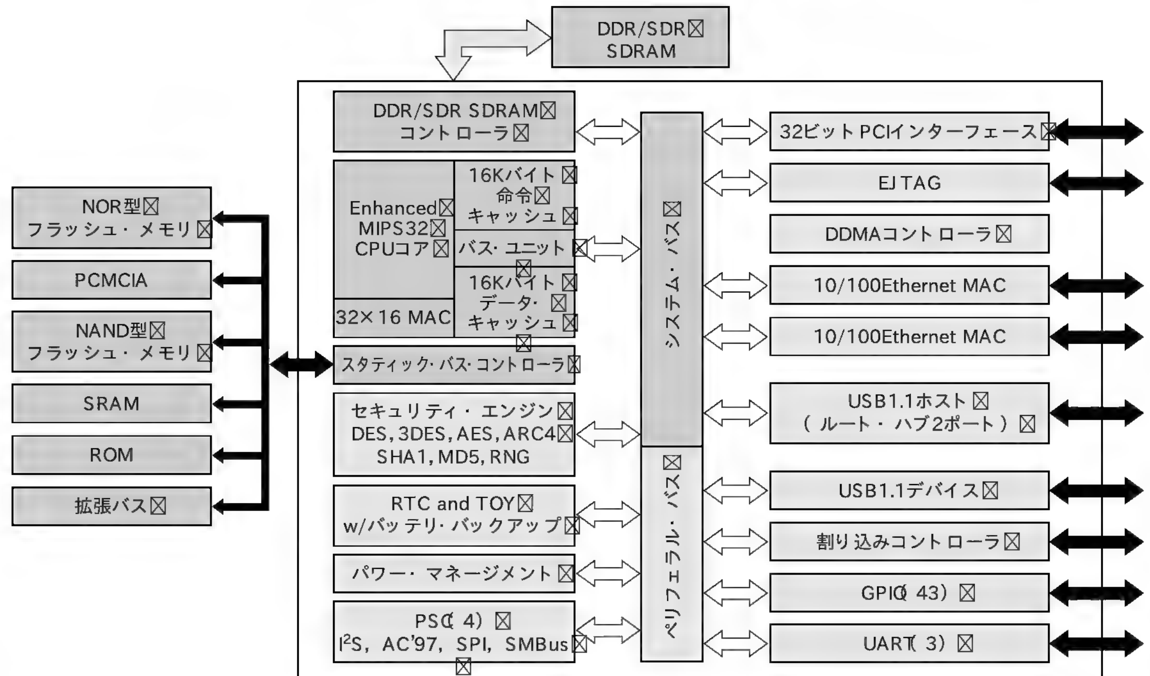


図4
Au1550のブロック図

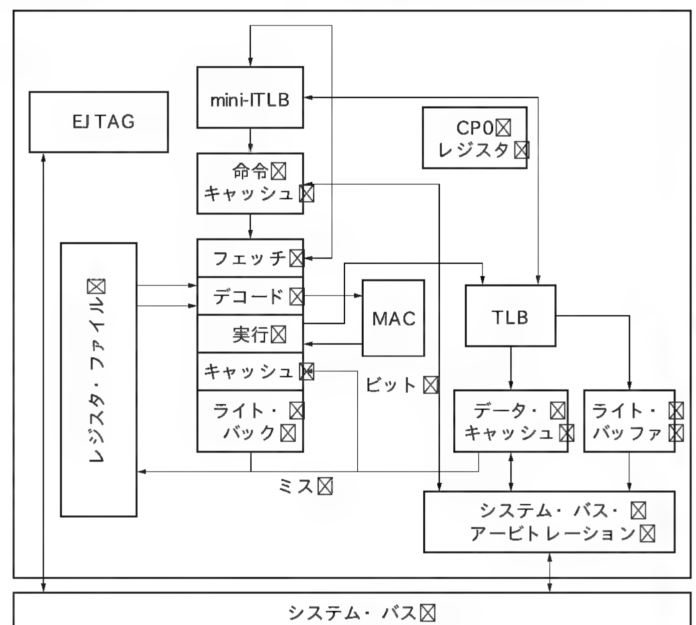


図5 Au1 CPUコアのアーキテクチャ

されます．

以下，ブロックごとに解説します．

● Au1 CPUコア

まず第一に，Alchemyソリューション SoCは，高速動作でありながら低消費電力であるということが最大の特徴です．

PCIバスを搭載したAu1500でさえ400MHz動作時に700mW(typ.)であり，ヒート・シンクすら使わずにシステムを作ることも可能です．モバイル製品向けに作られたAu1100では，400MHz動作時に250mW(typ.)であり，おそらくこのクラスのプロセッサでは最小の消費電力ではないでしょうか．実際，Au1100を使用したPDA型のデモ機では，MPEGの動画再生を連続4時間行うことができます．

では，MIPSアーキテクチャでありながら，なぜこのような低消費電力を実現できたのでしょうか．

その答えは，CPUコアの設計方法にあります．詳細は割愛しますが，ほかのMIPS CPUとの大きな違いは，CPUコアがAMD(旧Alchemy Semiconductor)の独自設計であるという点です．

MIPS CPUコアは，MIPS Technologies社からライセンスされています．たいていのMIPS CPUは，MIPS Technologies社から提供されているソフト・コア，あるいはハード・コアを利用し，独自の改良を加えるなどして作られています．

一方，AMDのAlchemyソリューション SoCでは，MIPS Technologies社から，MIPS32アーキテクチャのライセンスを受け，ゼロから独自にMIPS32のCPUコアを開発しました．これがAu1xxxに搭載されているAu1 CPUコアです．

Au1 CPUコアには，動作していないロジックに対してクロックを停止するといった独自の省電力制御技術が盛り込まれています．また，いわば職人技ともいえる手作業による回路レイアウトも高速動作と超低消費電力の両立に大きく寄与しています．

余談ですが，Au1 CPUコアを開発したチームは，かつてDECの半導体事業部においてAlphaやStrongARMを開発したエンジニアが中心になっており，高速動作や超低消費電力の設計に非常に精通し，また多くの実績をもっています．そのため，Alchemyソリューション SoCでは，すべてのAu1xxxデ

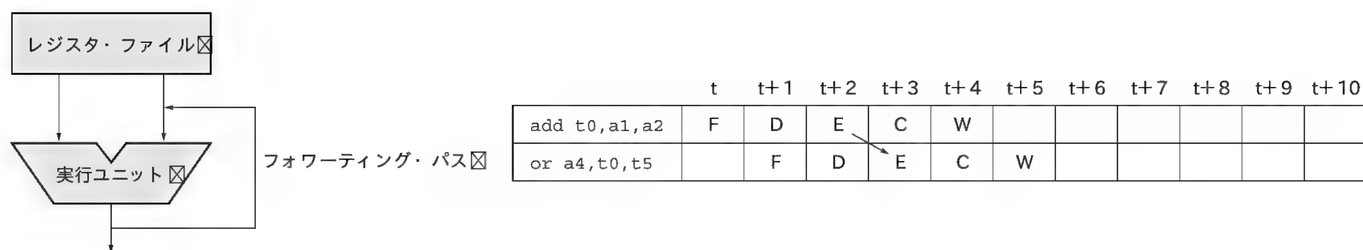


図6 リザルト・フォワーディング



図7 分岐命令の実行

バイスにおいてファースト・シリコンで目標の消費電力と動作スピードをほぼ実現し，同時にOSを起動できるほどの完成度にしあがりました．

Au1 CPUコアは，シンプルな5段パイプラインのスカラ・アーキテクチャを採用しています(図5，p.91)．昨今の高性能なプロセッサでは当たり前になっているスーパ・スカラや，スーパ・パイプラインを用いず，あえてスカラ・アーキテクチャにした理由は消費電力を抑えるためです．

当然のことですが，複雑なアーキテクチャには余分なロジックが必要になるので，どうしても消費電力が大きくなりがちです．通常，パイプラインの段数が少ないと，動作周波数を上げることが困難になりますが，Au1 CPUコアでは1次キャッシュ・アクセスのためのアドレス生成のタイミングを早めるといったアーキテクチャ上のくふうや，速いタイミングが必要な回路のトランジスタを大きくし，遅くても良い信号の出力トランジスタは小さくするなどの回路設計やレイアウト上でのくふうによって，0.18μmプロセスを使いながら500MHzの最高動作周波数を実現しています．

キャッシュは，命令，データともに16Kバイトずつで，これも最近の高性能なプロセッサとしては小さめなのですが，消費電力やコストを考えた際，16Kバイトは非常にバランスの良いサイズなのです．むやみに大きいサイズのキャッシュを搭載しても，ヒット率の上昇によるメリットより，コストや消費電力の増大によるデメリットのほうが大きくなってしまいます．

Au1 CPUコアは，シンプルな5段パイプライン(F：フェッチ，D：デコード，E：実行，C：キャッシュ，W：ライト・バック)ですが，決して旧来の伝統的なMIPSの5段パイプラインではありません．前述のとおり，分岐やロード/ストアのアドレス計算などは，ステージをまたいで処理を行います．また，直前の命令の実行結果を後続の命令がオペランドとして利用している場合，リザルト・フォワーディングによってパイプラインがストールすることを防ぎます(図6)．

分岐先の命令フェッチは実行ステージから始まるので，ディレイド・スロットは一つです(図7)．ディレイド・スロットに位置する命令(ディレイド・インストラクション)は，直前の条件分岐が分岐する/しないにかかわらず実行されます．したがって，ここに有効な命令を入れることができれば，分岐によるペナルティは0です．たとえNOPしか入れられなかったとしても，ペナルティは1命令だけなので分岐予測などの複雑なロジックをもつ必要がありません．

実行ステージで処理される命令は，そのほとんどが1クロックで処理を終えます．CLZやMULといった複数のサイクルが必要な命令によってレジスタへのライト・バックが遅れる際には，自動的にインタロックをかけるので，ライト・バック・ステージのスケジュールを意識してNOP命令を入れる必要はありません(図8)．

図8に，乗算命令によるパイプラインのインタロックの例を二つ示します．

mul命令はメイン・パイプラインの実行ユニットで処理され，結果はレジスタ・ファイルに入ります．実行に2クロック必要なので，その結果，t0を使う後続のadd命令は1クロックだけストールします．一方，mult命令は，メイン・パイプラインとは独立した積和演算器(MACユニット)で行います．結果はレジスタ・ファイルではなく，HI/LOレジスタに格納されます．後続のmflo命令はLOレジスタから積の下位32ビットを取り出す命令ですが，MACユニットでの処理が終了するまでパイプラインがストールします．

もちろん，このような場合，二つの命令の間にデータの依存関係がない命令を入れることでストールを防ぐこともできます．通常はコンパイラがオプティマイザによってコードを入れ換え，ストールの発生はある程度回避されます．

ちなみに，Au1 CPUコアのMACユニットは，MUL命令以外のすべての乗除算を行います．16×16，32×16ビットの乗算は1クロックで終了します．32×32ビットの乗算は，結果を汎用レジスタに得る場合は2クロックで終了します．2クロック必要ですが，1クロックごとに次の32×32ビット乗算を開

図8 インタロック

	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9	t+10
mul t0,t1,t2	F	D	E	C	W						
add v0,t0,t7		F	D	stall	E	C	W				
mul t0,t1	F	D	E	C	W						
mflo vo		F	D	stall	stall	stall	stall	E	C	W	

始できるので，連続して行う場合には，見かけ上，1クロックごとに32×32ビット乗算を行うことになります．結果をHI/LOレジスタに得る場合，16×16，32×16ビットの乗算は3クロックで，32×32ビットの乗算は4クロックで処理されます．除算は最大35クロック必要です．

Au1 CPUコアには浮動小数点演算器が内蔵されていないので，浮動小数点命令を実行しようとした場合，例外が発生します．

話が前後しますが，フェッチ・ステージでメイン・パイプラインに取り込まれる命令は，仮想アドレスで指定されるので，TLBによるアドレス変換が行われます．Au1 CPUコアは4エントリのインストラクションTLBを持っており，TLBミスヒットによる性能の低下を抑えています．

インストラクションTLBは命令フェッチ専用のTLBであり，ソフトウェアからは不可視です．また，意識的にプログラミングする必要もありません．メインのTLBに関しては後述します．

以上のようなAu1 CPUコアそのものに関するくふうや，後述するシステム全般の性能向上に関するくふうにより，Au1 CPUコアは，スカラ・プロセッサでありながら，Dhrystoneベンチマークにおいて，400MHz動作時に440MIPSという動作周波数に対して10%高い性能を得ています．

● キャッシュ

キャッシュ・アクセス・ステージでは，メモリへのアクセスが行われます．キャッシュにヒットすれば1クロックで処理は終了します．

もし，ミスヒットした場合，アクセスがロードかストアかで処理が分かれます．ストア動作の場合はライト・バッファに対して書き込み要求をポストし，1クロックで処理は終了します．ロードの場合，新しいキャッシュ・ラインがアロケートされ，システム・バスに対してデータのロードを要求します．その際に要求されるデータは，キャッシュ・ラインの先頭からではなく，実行中の命令が必要としているデータを最初に要求します．

したがって，最初のデータをロードした段階で即座にパイプラインは動作し始め，ミスヒットによる遅延を最小限に抑えます．

ただし，Au1500とAu1550にインプリメントされた，CCA=4に設定されたアドレス空間ではこの限りではなく，必ずキャッシュ・ラインの先頭からアクセスが発生します．この設定は，PCIバス上のメモリやデバイスにアクセスする場合に必要になります．

Normal	ヒット	ミス
loadまたは命令フェッチ(CCA＝3，4，5)	・キャッシュからパイプラインへデータを送る	・新しいラインをアロケートし，ライン全体をフィルする ・Lockビットをクリアする ・パイプラインへデータを送る
store(CCA＝3，4，5)	・パイプラインから受け取ったデータをキャッシュへ書き込む ・Dirtyビットをセットする	・パイプラインから受け取ったデータをライト・バッファに送る
Streaming	**ヒット**	**ミス**
loadまたは命令フェッチ(CCA＝6)	・キャッシュからパイプラインへデータを送る	・新しいラインをウェイ0にアロケートし，ライン全体をフィルする ・Lockビットは変更しない ・パイプラインへデータを送る
store(CCA＝6)	・パイプラインから受け取ったデータをキャッシュへ書き込む ・Dirtyビットをセットする	・パイプラインから受け取ったデータをライト・バッファに送る
store(0x4hint)	・何もしない	・新しいラインをウェイ0にアロケートし，ライン全体をフィルする ・Lockビットは変更しない
Lock	**ヒット**	**ミス**
CHCHE 0x1D/0x1C	・Lockビットをセットする	・新しいラインをアロケートし，ライン全体をフィルする ・Lockビットをセットする

CCA： Cache Coherency Attributes

図9 キャッシュの動作

新しいキャッシュ・ラインをアロケートする際，キャッシュに空きがない場合には，LRUアルゴリズムに従って適当なラインがフラッシュされます．キャッシュのヒット/ミスヒットや設定の違いによって，どのような動作になるかを図9に示します．

ポイントは，CCA=6，ストリーミング用に設定したキャッシュ・ラインの動作です．CCA=6に設定されたアドレス空間では，ロードやプリフェッチでミスヒットした際に，必ずウェイ0が使われます．通常，ストリーミング・データは再利用されることが少なく，かつサイズが大きいので，ストリーミング・データによってデータ・キャッシュ全体が使われてしまうと，システムの性能が低下してしまいます．CCA=6は，このような状態を防ぐために使用します．

Dirtyビットがセットされたキャッシュのアドレスは，システム・バス上でスヌープされ，コヒーレンシが保たれます．もしも，内蔵しているEthernet MACなどがメモリからデータをロードしようとしているときに，そのアドレスがデータ・キャッシュ内でDirtyビットがセットされた状態である場合，データ

はメモリからではなく，データ・キャッシュからシステム・バスに供給されます．

DMA 転送を行うすべての内蔵ペリフェラルは，システム・バスに対してコヒーレンシを維持するかしないかを指定し，アクセス要求を出すことができます．システム・バス上でキャッシュのコヒーレンシが保たれるため，デバイス・ドライバはEthernet MACなどの送信用バッファにデータを書き込んだ後，メモリに対してデータ・キャッシュをフラッシュさせる必要がなく，非常に効率良く，また高速に動作することができます．

● ライト・バッファ

STORE 命令がキャッシュにミスヒットしたり，キャッシュ可能な空間でないアドレス（CCA=2，7）に書き込もうとした場合，Au1 CPUコアはライト・バッファに書き込み要求をポストします（図10）．ライト・バッファは次の四つの効果で性能向上に寄与します．

1) メイン・パイプラインのキャッシュ・ステージを1サイクルで終了させることができ，キャッシュ・ミスによるパイプラインのストールを防ぐ
2) 複数のストア要求を保持することができ，パイプラインがストールする頻度を著しく減らす
3) 同じアドレスの32ビット・ワード内への複数のハーフ・ワード/バイト・データ書き込みをマージして，一つのワード・アクセスにすることができる
4) 連続するアドレスに対する単一ワードの書き込みを，バースト転送にして書き出すことができる

メイン・パイプラインのキャッシュ・ステージでポストされた書き込みデータは，マージ・ラッチに入ります．次にポストされたデータのアドレスが，マージ・ラッチ内のデータと同じアドレスである場合，二つのデータはマージ・ラッチ内で合成されます．マージ・ラッチ内のデータと違うアドレスのデータがポストされた場合，マージ・ラッチ内のデータが16エントリのFIFOに書き出され，ポストされたデータがマージ・ラッチに入ります．マージ・ラッチによって，ストリング処理やメモリ・コピー時のバス・アクセスを激減させることができます．

ライト・バッファやマージ・ラッチの動作は，アプリケーション・ソフトウェアで意識する必要はありませんが，カーネルやドライバを書く場合には注意が必要です．

ストアされたデータはライト・バッファに入り，FIFOがいっぱいになるまで実際には書き出されません．ペリフェラルのレジスタを設定したり，送信レジスタにデータを書き込もうとしているような場合，ストアを実行した段階では，実際にはペリフェラルに対して書き込みがされないことになります．このような場合，SYNC 命令を実行することによって強制的にライト・バッファをすべてフラッシュし，ペリフェラルに対しての書き込みを実行させることができます．

また，ライトバッファ内のデータはスヌープされません．しかしながら，通常はドライバを書く際，まずバッファにデータを用意した後，ペリフェラルに対して送信指示を出します．この手順を守り，最後にSYNC 命令を実行すればコヒーレンシを保つことができます．

逆に，ライト・バッファ内にあるデータのアドレスから，メイン・パイプラインがロードを行おうとした場合はどうでしょうか．この場合，ロードしようとしているデータが格納されているFIFOのエントリまで，データがフラッシュされます．メイン・パイプラインは，必要なデータがフラッシュされた後，システム・バスからそのデータを取り込むので，コヒーレンシは保たれます．

● TLB

Au1 CPUコアは32組のデュアル・エントリ，フルセット・アソシアティブTLBを内蔵しています．このTLBは，MIPS32で規定されたTLBそのものであり，独自の仕様ではありません（図11）．PageMaskの設定により4K，16K，64K，256K，1M，4M，16Mバイトのページ・サイズをサポートします．EntryLo0/1に設定される物理アドレスは，36ビットの空間をもっています．リセット直後，TLBは不定の値が入っているので，適宜初期化しなければなりません．簡単な初期化プログラムをリスト1に示します．

● Coprocessor 0

MIPS32プロセッサのCoprocessor 0（CP0）は，多くがMIPSによって定義されていますが，オプションでインプリメントがデバイス・ベンダに任されている部分や，独自のインプリメントが許されている部分があります．Au1 CPUコアで独自にインプリメントしたレジスタに関しては，Au1xxxデータ・ブックを参照してください．

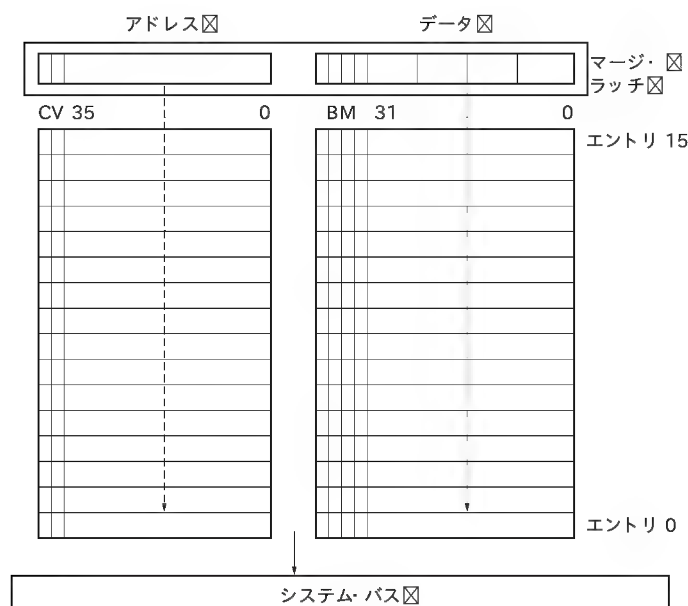


図10　ライト・バッファ

PageMask	0	PageMask	0			
EntryHi	VPN2	0	ASID			
EntryLo0	0	PFN0	CCA	D	V	G
EntryLo1	0	PFN1	CCA	D	V	G

(ビット位置：31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00)

図11　Au1 CPUコアのTLB

2 Alchemyシリーズ内蔵周辺コントローラ

●システム・バス（SBUS）

Au1xxxは，非常に高速かつ効率の良いシステム・バスをもっています．通常，システム・バスはCPU_PLLの1/2の速度で動作するので，400MHzのAu1xxxではシステム・バスのスピードは200MHzになります．sys_powerctr（SD）レジスタの設定により，CPU_PLLの1/3や1/4に設定することもできます．また，sys_powerctr（SB）により，システム・バスが使われていないときには，sys_powerctr（SD）で設定した周波数の，さらに1/2に落とすことができ，省電力化に貢献します．

図12に，Au1550を例にしたシステム・バスにつながるさまざまなユニットを示します．Au1 CPUコアは当然のことながら，高速なデータ転送が要求されるペリフェラルや，二つのメモリ・コントローラ，およびDMAコントローラなどがシステム・バスに直接つながっています．高速性がそれほど要求されないペリフェラル類は，ペリフェラル・バスを通してシステム・バスにつながります．

システム・バスに直接つながっているバス・マスタからのバ

リスト1　簡単な初期化プログラム

```
        .global asmTlbInit
asmTlbInit:
        li   t0, 0              # index value
        li   t1, 0x00000000     # entryhi value
        li   t2, 32             # 32 entries
tlbloop:
        /* Probe TLB for matching EntryHi */
        mtc0 t1, CP0_EntryHi
        tlbp
        nop

        /* Examine Index[P], 1=no matching entry */
        mfc0 t3, CP0_Index
        li   t4, 0x80000000
        and  t3, t4, t3
        addiu     t1, t1, 1     # increment t1 (asid)
        beq  zero, t3, tlbloop
        nop

        /* Initialize the TLB entry */
        mtc0 t0, CP0_Index
        mtc0 zero, CP0_EntryLo0
        mtc0 zero, CP0_EntryLo1
        mtc0 zero, CP0_PageMask
        tlbwi

        /* Do it again */
        addiu     t0, t0, 1
        bne  t0, t2, tlbloop
        nop

        /* Establish Wired (and Random) */
        mtc0 zero, CP0_Wired
        nop
        jra
        nop
```

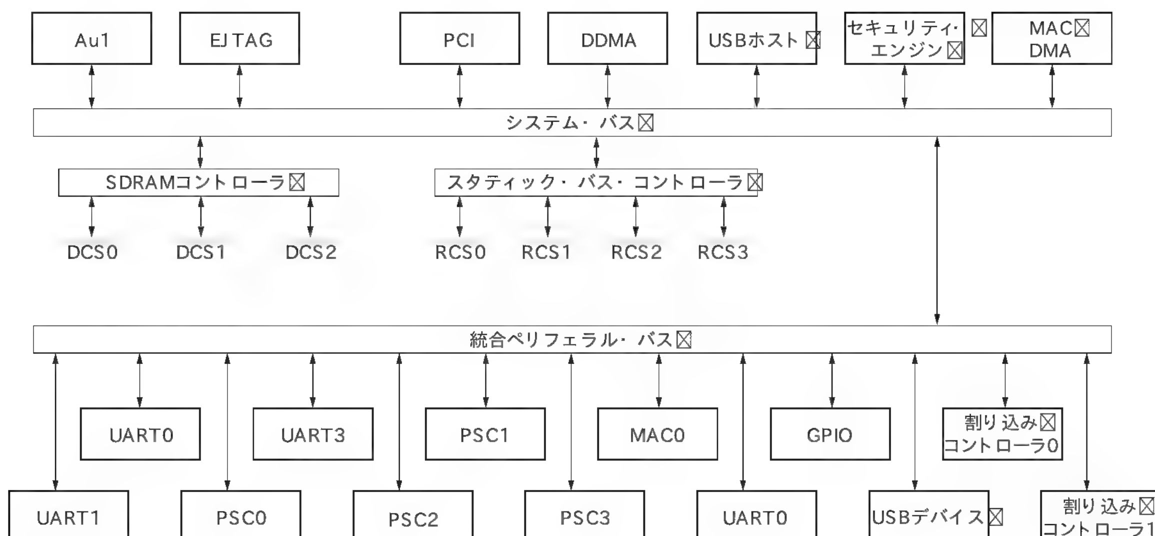


図12　Au1550のシステム・バス

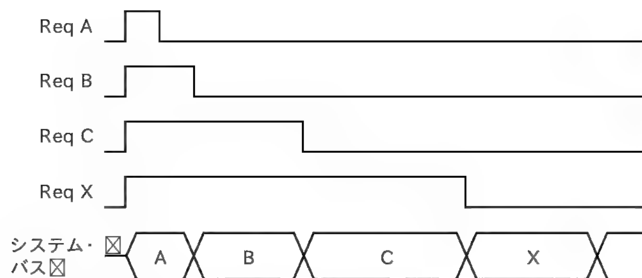


図13　システム・バスのアービトレーション

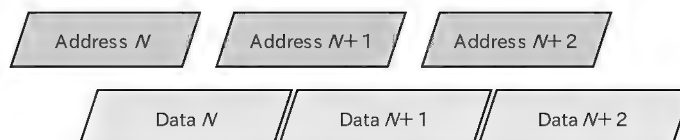


図14　システム・バスの動作

ス・リクエストは，ラウンドロビンで処理されます（図13）．ペリフェラル・バスにつながったユニットからのバス・リクエストは，ペリフェラル・バス内でラウンドロビンで処理され，一つのシステム・バス・リクエストに集約されます．

システム・バス上でのデータ転送は，アドレッシングのフェーズとデータ転送のフェーズに分かれますが，両者は可能な限りオーバラップして実行されます（図14）．そのため，高速の動作クロックと相まって，非常に高いスループットを実現しています．

● SDRAMコントローラ

Au1xxxのもつSDRAMコントローラは，最大3バンクのSDRAM，SMROM（Synchronous Mask ROM），およびシンクロナス・フラッシュ・メモリを直結することができます．

Au1550では，さらにDDR200，DDR266，DDR333，およびDDR400 SDRAMもサポートします．ここでいうバンクとは，SDRAMデバイス内のバンクのことではなく，Au1xxxのSDRAMインターフェースに接続されたSDRAMデバイスの組み数のことです．

Au1000，Au1100，Au1500では，つねに32ビット・バス幅で接続しなければならないので，一般的な×16ビット・バスのSDRAMを使用した場合，2個で1バンクになります．したがって，最大6個のSDRAMデバイスを直接ドライブできるということになります．

Au1550では，16ビット・バス幅でSDRAMを使用することもできるので，1バンク1個で構成することもできます．SDRAMデバイス内のバンク・サイズは，2または4バンクをサポートします．アドレス線はAu1000，Au1100，Au1500で13ビット，Au1550では14ビットあるので，将来的にメモリ・デバイスの容量が増えた場合でも十分に対応できます．

SDRAMバスの動作周波数は，システム・バスの1/2になります．したがって，CPU_PLLの1/4にした場合，400MHzのAu1xxxでは100MHzになります．同様に333MHz品では83MHz，500MHz品では125MHzになります．Au1550ではSDRAMバスの動作周波数をシステム・バスと同じに設定できるオプションが追加されました．この機能を使うことで，400MHz品でDDR400（クロックは200MHz）を利用することができ，ネットワーク機器などの高いデータ・スループットが必要なアプリケーションに対応できます．また，DDR400を16ビットで接続することにより，スループットをある程度確保しながらシステム・コストを下げることも可能です．

● スタティック・バス・コントローラ

Au1xxxは，SDRAMコントローラとは別に，スタティック・メモリをつなぐためのスタティック・バス・コントローラをもっています．SDRAMコントローラとスタティック・バス・コントローラが独立して動作するため，SDRAMへのデータの書き出しと同時にフラッシュ・メモリからのデータの読み書き，またはその逆が可能であり，システム性能の向上に貢献しています．

Au1xxxのスタティック・バス・コントローラは，ROM，SRAM，NOR型フラッシュ・メモリ，PCMCIAカード，CFカード，IDEハードディスク・ドライブを直結することができます．

Au1550では，さらにNAND型フラッシュ・メモリにも対応しています．また，外付けのペリフェラルを利用する際にも，拡張バスとして動作します．バス幅は16ビット，または32ビットに設定できます．動作クロックはSDRAMコントローラと同様に，システム・バスの1/2になります．

スタティック・バス・コントローラでは，最大で四つのチップ・セレクト信号を利用することができ，それぞれにSRAM，I/Oデバイス，PCMCIA/CF/IDE，NOR型フラッシュ・メモリ，NAND型フラッシュ・メモリの動作モードを設定できます．ただし，PCMCIA/CF/IDEモードは一つしか選択できません．また，このモードでは，バス幅が16ビットに固定されます．残念ながら，Card Busとして利用することはできません．

PCMCIAカードやCFカードを2組接続したり，活線挿抜に対応する場合には外付けのバッファが必要です．また，一部の信号はGPIOを使って代用します．

● 内蔵ペリフェラル

Au1xxxには，さまざまなペリフェラルが内蔵されており，多くのアプリケーションでは，メモリをつなぐだけでシステムを構築できます．Au1 CPUコア，システム・バス，SDRAMおよびスタティック・バス・コントローラはAMDの独自設計ですが，内蔵ペリフェラルの多くは外部ベンダのIPを利用しています．

実績のあるIPを利用することで，Ethernet MACやUSBコントローラなどは，ソフトウェアの互換性やハードウェアそのものの信頼性を高めることができます．他社のIPを利用して

はいますが，省電力化の対策のため，すべてのペリフェラルは，未使用時にはクロックを停止することができるようになっています．誌面のつごう上，すべてのペリフェラルについて解説することはできないので，いくつかの特徴的なものと，注意が必要なものを解説します．

また，各ペリフェラルの初期化に関しては，サンプル・プログラムを解説する際に紹介します．

▶ 10/100 Ethernet MAC

Au1xxxが内蔵するEthernet MACは，MIIインターフェースによって外付けのPHYと接続されます．Ethernet/IEEE 802.3スペックのプロトコルをすべてサポートし，Au1 CPUコアの負荷を減らすための，さまざまな機能をもっています．高速なシステム・バスに直結されているため，Au1000，Au1500，Au1550においては内蔵している二つのEthernet MAC間でIPフォワーディングを行い，100Mbpsのライン・スピードを飽和させることができます．

また，NAT程度の処理であれば，Au1 CPUコアで処理しながら90Mbps以上のスピードを確保できます．

Au1xxxが内蔵するEthernet MACは，専用のDMAコントローラをもっています．一つのEthernet MACに送信用と受信用として，二つのDMACをもっています．各DMACには4エントリのリング・バッファを設定し，順番に転送を行うように作られています(図15)．一つのエントリのバッファに対する送/受信が終了すると，割り込みとステータス・ビットによってそれを知らせ，自動的に次のエントリの処理に移行します．

Ethernet MACのプログラミングに関しては，Au1xxxのデータ・ブックで詳しく解説されています．

▶ USBコントローラ

Au1xxxは，USBホスト・コントローラとUSBターゲット・コントローラを一つずつ内蔵しています．これらのUSBコントローラは，USB1.1，およびOHCIインターフェース規格です．OHCI規格に定められたすべての割り込みをサポートしています．Au1550においては，ホスト・コントローラとターゲット・コントローラを組み合わせてOn-The-Goの機能をもたせることもできます．

▶ カウンタ/タイマ

Au1xxxは，TOY(Time Of Year)，RTC(Real Time Clock)という二つのカウンタ/タイマをもっています．Au1xxxの資料では，これらはリアルタイム・クロックと呼ばれていますが，一般的にいうリアルタイム・クロックとは若干異なるので，ここではあえてカウンタ/タイマと記述します．

TOY，RTCは，両方とも32KHzのクロックでカウントを繰り返します．設定した値までカウントすると，割り込みを発生したり，スリープ・モードから復帰させることができます．

スリープ・モード中は，TOYのみカウントが継続するので，一般的にいわれるリアルタイム・クロックとして使用することも可能です．後述しますが，スリープ・モードではV_{DDX}という3.3VのI/O用の電源をONにし続けなければなりません．そのため，バッテリ・バックアップがしにくいという欠点があり，真にリアルタイム・クロックであると言い切ることができません．

Au1550では上記欠点を改良し，TOY, RTC用の電源ピンを別に設け，ボタン電池でバックアップできるようになっています．

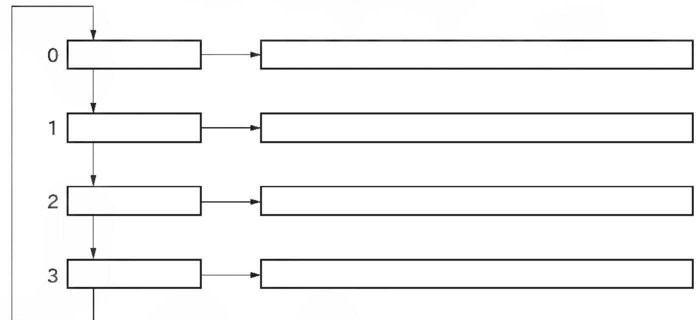


図15 Ethernet MACリング・バッファ

表1 LCDコントローラの特徴

サポートするLCDパネルの種類	4/8 bit mono single passive matrix STN panels
	8 bit color single passive matrix STN panels
	16 bit color dual passive matrix STN panels
	12/16 bit TFT panels
	18 bit TFT panels(up to 65,536 colors)
	Panel sizes up to 800×600 are supported
ディスプレイ・モード	1/2/4/8 bpp palletized TFT
	12/16 bpp non-palletized TFT
	1/2/4 bpp mono STN
	1/2/4/8 bpp palletized color STN
	12 bpp non-palletized color STN
その他の機能	ダブル・バッファのサポート
	320×240以下の解像度では，ハードウェアでの90°，180°，270°回転表示が可能
	輝度，コントラスト調整用のPWMを2組内蔵

▶ LCDコントローラ(Au1100)

Au1100が搭載しているLCDコントローラの特徴は，表1に示すとおりです．

このLCDコントローラのアーキテクチャは，ビデオ・メモリをメイン・メモリ(SDRAM)内に確保するUMAです．したがって，解像度が大きいほどビデオ・リフレッシュのためにシステム・バスが占有される率が大きくなり，システム性能が低くなることを考慮しなければなりません．しかしながら，SVGA表示の画面に，QVGAサイズのMPEG-1のビデオを再生するくらいの性能は確保できます．

LCDコントローラは，ディスプレイを安定してドライブし続けなければならないので，システム・バスに対して高いプライオリティが割り当てられています．内蔵ペリフェラルを使用している限り，それらがシステム・バスを占有する期間は最大8ワードのバースト転送なので問題はありません．

それよりも重要なのは，外部に接続されたペリフェラルとの

システム・バスの共有です．スタティック・バスに接続された外部ペリフェラル，特にPCMCIAデバイスが長期間WAITをアサートすると，その間はシステム・バスもウェイト状態になってしまいます．その結果として，ビデオ・リフレッシュが間に合わず，画面が乱れてしまうことになります．これを回避する指針として，“Au1100 Processor LCD Performance”というアプリケーション・ノートがあります(AMD Webサイト参照)．

▶ PCI Rev.2.2コントローラ(Au1500/Au1550)

Au1500，Au1550にはPCIコントローラが搭載されており，表2に示す特徴をもっています．

PCIのメモリ・マップは，図16のようになっています．見てわかるように36ビット物理アドレスの上位4ビットが0ではないので，kseg0，kseg1空間としてアクセスすることができません．したがって，TLBの設定が必須となります．TLBを設定するサンプル・プログラムをリスト2に示します．

▶ セキュリティ・エンジン(Au1550)

Au1550が内蔵しているセキュリティ・エンジンには，業界でもっとも実績のあるSafeNetのIPを利用しています(図17)．このエンジンは以下に示すハードウェア機能をもっています．

表2 PCI 2.2コントローラの特徴(Au1500，Au1550)

- PCI Rev.2.2に準拠
- 最高66MHzまで任意のクロックで動作，外部からのクロック供給も可能
- 32ビット・データ・バス
- 3.3Vのみサポート
- アービタは最大四つのバス・マスタをサポート
- 外部バス・マスタからAu1500/Au1550のメモリ空間をアクセス可能
- 外部のアービタを利用することも可能
- ホスト・モード/サテライト・モードどちらでも動作可能

物理アドレス⊠	機　能⊠
0x0140050xx	Au1550 コンフィグレーション・レジスタ⊠
0x0140051xx	Au1550 PCI ヘッダ空間⊠
0x4xxxxxxxx	PCI メモリ空間⊠
0x5xxxxxxxx	PCI I/O 空間⊠
0x6xxxxxxxx	PCI 外部コンフィグレーション空間⊠

図16 PCIメモリ・マップ

リスト2 TLBを設定するサンプル・プログラム

```
このリストは，//YAMON/arch/pci/arch_pci_pb1500.c からの抜粋です．

#include <sysdefs.h>
#include <syserror.h>
#include <syscon_api.h>
#include <sys_api.h>
#include <syserror.h>
#include <pb1000.h>
#include <pci_api.h>
#include <arch_pci_pb1500.h>

#include <stdio.h>

void     sys_tlb_write();
UINT32   CP0_prid_read();
void     CP0_wired_write();

/*
 *  arch_pci_pb1500.c
 *
 *  pb1500 PCI initialization
 *
 *  Copyright 2001, Alchemy Semiconductor, Inc.
 */

/*****************************************************************
 *                          pci_init
 *  Description :
 *  -------------
 *
 *  PCI module initialization function
 *
 *  Uses the first three entries of the MMU to map
 *                                the following 32 MB areas:
 *
 *      0    0x40000000 - 0x41ffffff      PCI non-cacheable
 *                                                 memory space
 *      1    0x50000000 - 0x51ffffff      PCI I/O space
 *      2                                 reserved for config
 *                                                       access
 *
 *  Return values :
 *  ---------------
 *
 *  OK if no error, else error code
 *
******************************************************************/

UINT32   pci_init(
              UINT32    mem_start,     // ignores these parameters
              UINT32    mem_size,
              UINT32    mem_offset,
              UINT32    io_start,
              UINT32    io_size,
              UINT32    io_offset
              )
{
    int       i;
    UINT32    pte[5];

    // NOTE: Au1500 PCI init code located into reset_pb1500.S

//  map physical address 0x400xxxxx to 0x40xxxxxx for PCI memory

    pte[0] = 0;                  // entry 0
    pte[1] = 0x01ffe000;         // 16MB pages for memory
    pte[2] = 0x40000000;         // logical VPN
    pte[3] = (0x40000000 >> 2)   // 0x4_0000_0000 physical address
              | 0x0017;    // non-cacheable, d, v, and g bits set
    pte[4] = (0x40100000 >> 2) // 0x4_0100_0000 physical address
              | 0x0017;    // non-cacheable, d, v, and g bits set
    sys_tlb_write(pte);

//  map physical address 0x500xxxxx to 0x50xxxxxx for PCI I/O

    pte[0] = 1;                  // entry 1
    pte[1] = 0x01ffe000;         // 16MB pages for I/O
    pte[2] = 0x50000000;         // logical VPN
    pte[3] = (0x50000000 >> 2)   // 0x5_0000_0000 physical address
              | 0x0017;     // non-cacheable, d, v, and g bits set
    pte[4] = (0x50100000 >> 2)   // 0x5_0100_0000 physical address
              | 0x0017;     // non-cacheable, d, v, and g bits set
    sys_tlb_write(pte);

    sys_tlb_read(0,pte);
    sys_tlb_read(1,pte);

    CP0_wired_write(4);      // keep 4 entries fixed

    return OK;
}
```

●DES，3DES，AES，ARC4エンクリプション
●MD5，SHA1ハッシュ処理
●IPSecパケットのヘッダ/トレイラ処理，パディング，イニシャライゼーション・ベクタ(IV)処理
●乱数発生機能

SafeNetのセキュリティ・エンジンを搭載しているので，CGXスクリプト・グラフィック・ライブラリや，QuickSecツール・キットを利用することができます．また，CGXライブラリ上で動作するソフトウェアは，そのままAu1550でも動作します．

IPSecパケットのヘッダ/トレイラ処理をハードウェアで行うので，非常に高速にIPSec処理をこなすことができます．また，ホワイト・ノイズを発生源にもつ乱数発生器を備えているので，ソフトウェアによる疑似乱数に比べて強固な暗号化を実現できます．

このセキュリティ・エンジンは，IPSec用に作られていますが，エンクリプション機能や，乱数発生機能は個別に利用できるので，マルチメディア処理に欠かせないディジタル・ライツ・マネージメント(DRM)に利用することもできます．

▶ プログラマブル・シリアル・コントローラ(Au1550)

Au1xxxは，AC97，I^{2}S，SPIなどのクロック同期型のシリアル・インターフェースは，個々に専用の信号ピンをもっていました．これらのインターフェースは，Au1550ではプログラマブル・シリアル・コントローラ(PSC)として一つのペリフェラル・ブロックにまとめられました(図18)．PSCは4組のシリアル・コントローラのユニットをもっており，それぞれにSPI，I^{2}S，AC97またはSMBusの機能を任意に割り当てることができます．

▶ ディスクリプタ・ベースDMAコントローラ(Au1550)

Au1550のDMAコントローラは，ほかのAu1xxxとは違ってディスクリプタ・ベースのDMAコントローラ(DDMA)になっています．従来のDMACでは，メモリ-メモリ間，ペリフェラル-ペリフェラル間での転送はできませんでした．また，二つのバッファ・エリアを交互に使って転送を行うことしかできなかったので，たとえばフレーム・バッファ内のウィンドウを移動するようなDMA転送には，ある程度ソフトウェアの介在が必要でした．

Au1550のDDMAではこのようなDMA転送を，一つのディスクリプタを記述することで，ソフトウェアが介在することなしに実行することができます．

Au1550のDDMAは16チャネルあり，内蔵ペリフェラルや，スタティック・バスに接続された外部ペリフェラルに割り当てることができます．ディスクリプタの記述により，メモリ-メモリ，メモリ-ペリフェラル，ペリフェラル-ペリフェラル間の転送を行うことができます．36ビットのソース・デスティネーション物理アドレスは，アライメントがずれていてもかまいません．また，転送中にビッグ・エンディアン/リトル・エンディアンの変換を行うこともできます．

一つのディスクリプタによって，矩形領域，ストライド・アクセス，不連続領域(scatter/gather)の転送を記述できます

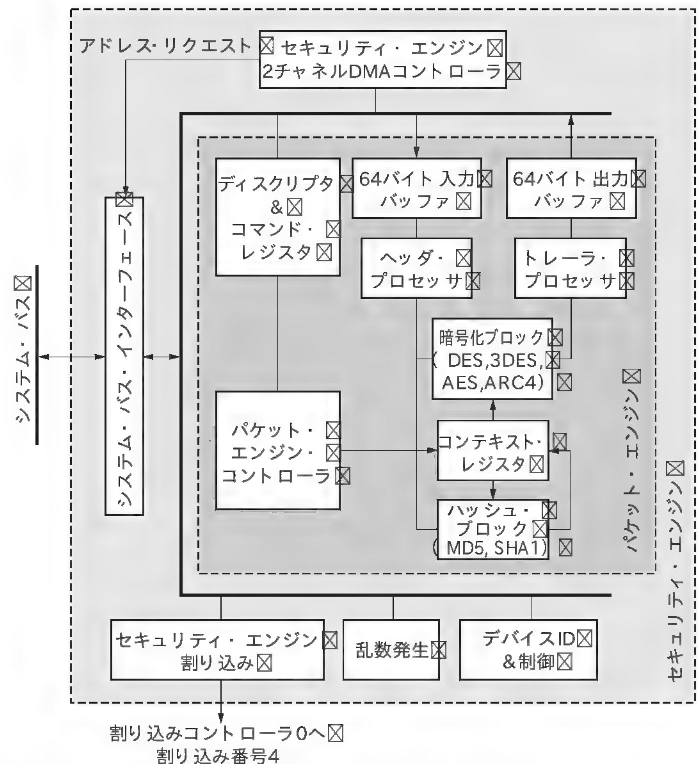


図17　Au1550のセキュリティ・エンジンのブロック図

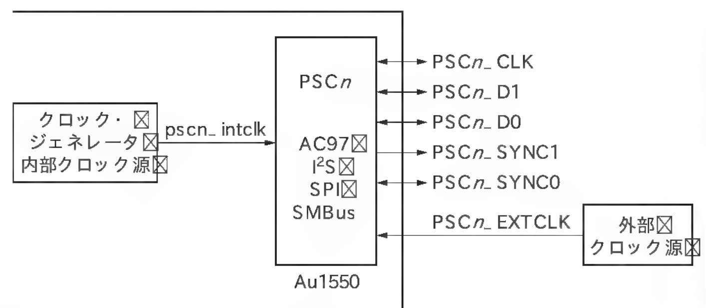


図18　Au1550のPSCのブロック図

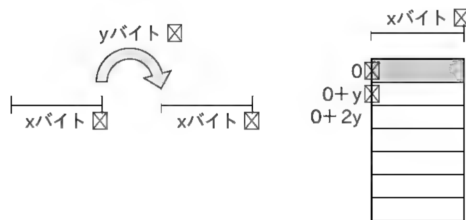
(図19)．また，ディスクリプタはカスケード接続することができ，一つのディスクリプタのDMAが終了した後，次のディスクリプタのDMAに移ることができます．

さらに，ディスクリプタをループしたり，条件分岐やサブルーチン化することも可能なので，非常に高度で複雑なDMA転送を行うことができます．

● パワー・マネージメント

Au1xxxは，IdleモードとSleepモードの2段階のパワー・セーブ・モードをもっています．Au1550ではそれに加えて，Hibernateモードをもっています．

さらに，Au1xxxは，CPUの動作周波数もダイナミックに変更することができます．図20に，Idle0/1とSleepモードへの入り方，および復帰の方法を示します．



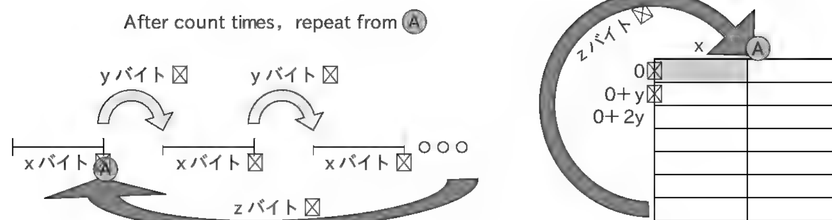


図19　DDMAの矩形領域/ストライド転送

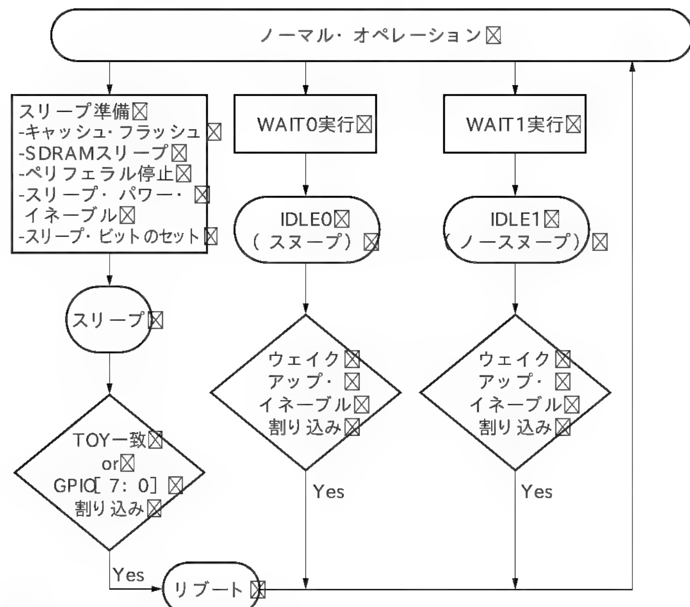


図20　パワー・セーブ・モード

リスト3　WAIT命令の手順

```
    .global au1_wait
au1_wait:
    la t0,au1_wait # obtain address of au1_wait
    cache 0x14,0(t0) # fill icache with first 8 insns
    cache 0x14,32(t0) # fill icache with next 8 insns
    sync
    nop
    wait 0
    nop
    nop
    nop
    nop
    j ra
```

▶ CPUクロックの変更

Au1xxxのCPU_PLLの変更は，簡単に行うことができます．sys_cpupllレジスタに，希望する値を設定するだけです．設定する値は，希望する周波数の1/12です．最小値は16(192MHz)で，最大値はCPUの最高速度を超えない値です．このレジスタを変更した場合，PLLの再同期のため，Au1xxxは約20μsの間，Au1550は約40μsの間だけ停止します．

▶ Idleモード

Idleモードは，内蔵ペリフェラルやシステム・バス，メモリ・コントローラなどを動作させたまま，Au1 CPUコアを停止するモードです．IdleモードにはIdle0，Idle1という二つのモードがあります．

Idle0は，Idle中システム・バスをスヌープし，ペリフェラルからのメモリ・アクセスにおけるデータ・キャッシュの利用を有効にします．

Idle1は，データ・キャッシュも含めて完全にAu1 CPUコアが停止するため，より消費電力が低くなります．

どちらのモードに入るかは，WAIT命令のオペランドによって決定します．Idleモードへは，WAIT命令の実行で入ることができますが，Idleモードに入る時点で命令フェッチをメモリから行っていると，Idleモードのメカニズムが正常に動作できなくなります．そのため，WAIT命令の手順を完全にキャッシュ内に取り込むために，リスト3に示す手順を用います．

Idleモード中は，sys_powerctr[SI]ビットをあらかじめ設定することによって，システム・バスのクロックを通常の1/2に減らすことができます．

Idleモードからの復帰には割り込みを用います．言い換えれば，どのような割り込みが入っても，Idleモードから復帰できます．Idleの最中には，汎用レジスタやCP0の内容は保存されます．しかしながら，CP0のCounterレジスタは，Idle中はカウントされません．したがって，Idle中はCounter/Compareをシステム・タイマ・ティックのような目的には利用できないことになります．

▶ Sleepモード

Sleepモードでは，Au1 CPUコアだけでなく，Au1xxx全体が省電力モードになります．Sleepモードに入るためには，いくつかの手順が必要です．Sleepモードに入ると，キャッシュ，TLBや，ほとんどのレジスタの内容は保存されないので，復帰後に必要な情報は適宜メモリに保存しておかなければなりません．Sleep中はTOYカウンタのみ動作し，ほかのクロックはすべて停止します．CPUコアは完全に停止し，1.1～1.5VのV_{DDI}はOFFにできます．しかしながら，TOYを動作させるため，I/O系の3.3V電源V_{DDX}は通電し続ける必要があります．Au1100とAu1550では，SDRAM用の電源V_{DDY}もOFFにでき

ます．Sleepモードでの消費電力は50μA以下です．

Sleepモードに入る手順を次に示します．Sleep中SDRAMは使用不能になるので，この手順はSDRAM以外のメモリから実行しなければなりません．

1) sys_slppwrレジスタに書き込み動作を行うことにより，デバイス内のSleep用電源回路を起動する
2) 使用しているすべてのペリフェラルを停止する
3) データ・キャッシュをすべてフラッシュし，インバリデートする
4) Sleepモード中でもSDRAMの内容を保存したい場合は，SDRAMをセルフ・リフレッシュ・モードにする．保存しなくて良い場合には，単にディセーブルする
5) GPIO[7:0]を復帰のために使用する場合，sys_pininputenをクリアする(システムの初期化のときに行っていれば，ここで再度行う必要はない)
6) 前回のSleep復帰の際，1にセットされたsys_wakesrcレジスタのビットをクリアする
7) 復帰に使用する割り込みのソースを，sys_wakemskレジスタに適宜設定する
8) sys_sleepレジスタに書き込み動作を行うことにより，Sleepモードに入る
9) Sleepモードに入ると，PWR_EN信号がネゲートされるので，必要に応じてV_{DDI}，V_{DDY}をOFFにする

また，Sleepからの復帰は，次に示す手順で行います．

1) Sleep復帰の条件に合致すると，PWR_EN信号がアサートされる．これを受けてsys_powerctr[VPUT]で設定した期間以内にV_{DDI}を投入しなければならない．Au1100，Au1550でV_{DDY}をOFFにしている場合，同様にONにする
2) プロセッサは，リセット・ベクタ0x1FC00000から命令フェッチを開始する
3) リセット・ベクタに置かれた初期化プログラムは，sys_wakesrcレジスタを調べ，Sleepからの復帰かどうかを判断し，適宜SDRAMからのデータの復帰などを行う
4) 通常のシステム初期化処理を行う

▶ Hibernateモード(Au1550)

Au1550では，さらに消費電力を低くできる，Hibernateモードが利用できます．Hibernateモードは，Sleepモードと同様に，TOYカウンタのみ動作し続けます．Hibernateモードは，SleepやIdleとは違い，ハードウェアによって$\overline{\text{WAKE}}$，$\overline{\text{FWTOY}}$信号で制御します．Sleepモードでは，V_{DDX}を通電し続けなければなりませんが，HibernateモードではそれをOFFにできます．TOYの動作に必要な電源は，XPWR32にボタン電池などをつないで供給します．

Hibernateモードに入るためには，$\overline{\text{FWTOY}}$信号をアサートします．Hibernateモードは，基本的にシステム全体の電源を落とすときに使用するモードなので，Sleepモードのようにデータ・キャッシュをフラッシュするといった前処理を行う必要はありません．ソフトウェアがどのような状態であっても，$\overline{\text{FWTOY}}$信号をアサートすることによってHibernateモードに入ります．

Hibernateモードからの復帰は，通常の電源立ち上げのシーケンスと同じです．Hibernateモードからの復帰であることは，sys_wakesrcレジスタを参照することで知ることができます．リセット・ベクタに置かれた初期化プログラムは，sys_wakesrcレジスタを調べ，Hibernateモードからの復帰かどうかを判断し，適宜初期化処理を行わなければなりません．

Hibernateモードの場合，TOYカウンタがsys_toymatch2レジスタと同じになった場合に$\overline{\text{WAKE}}$信号をアサートすることができます．外部ロジックから$\overline{\text{WAKE}}$信号のアサートを検出して電源を立ち上げることにより，アラーム・クロックの機能を実現できます．

3 ブート処理

リセット後のAu1xxxの初期化について解説します．Au1xxxの初期化は，おおむね次の手順で行います．

(1) CPUエンディアン・モードの設定

通常のシステムにおいては，CPUのエンディアン・モードは初期化手順の早い時期に設定することが重要で，たいていの場合は最初に行う処理になります．Au1xxxではソフトウェアによってエンディアン・モードを変更できますが，リセット後のデフォルトはビッグ・エンディアンです．

エンディアン・モードはブートROMの内容の解釈に影響します．32ビット・ワード(たとえば命令コードなど)は，リトル・エンディアンでもビッグ・エンディアンでも同様に扱われますが，8ビット，16ビット・データは影響を受けます．

したがって，データのアクセスを始める前にエンディアン・モードを設定しなければなりません．アプリケーションをコンパイルする際にも，同じエンディアン・モードでコンパイルしておかなければなりません．

(2) ステータス・レジスタの設定

プロセッサの動作モードを決めるため，ステータス・レジスタを適宜設定しなければなりません．

多くの場合，Status[BEV]は1(エクセプション・ベクタ・テーブルをkseg1領域に設定するため)にセットし，Status[ERL]とStatus[EXL]は"normal"動作に，そしてブート中の割り込みを禁止するためにStatus[IM]とStatus[IE]はクリアします．これらの設定は，CP0に0x00400000を書き込むことによって行うことができます．

(3) Config0の設定

CP0 Config0レジスタのK0フィールドは，kseg0領域のキャッシュ・コヒーレンシ属性を決定します．このフィールドはデフォルトでCCA＝3，キャッシュ可能＋コヒーレントに設定されますが，システムに最適なモードに設定し直すことができます．

（4）Watch機能を無効にする

ブート処理の間，不要なwatchpoint例外が発生しないよう，Watchおよび IWatch機能を無効にしておかなければなりません．CP0のWatchLoと IWatchLoに0を書き込むことにより，watchpoint関係の機能を無効にできます．

（5）Performance Counterを無効にする

デバッグ中や，性能プロファイルの測定中以外では，Performance Counterは必要ありません．Performance Counterを無効にすることで，消費電力を減らすことができます．

（6）EJTAGデバッグ・レジスタの設定

プロセッサを正常に動作させるためには，EJTAGデバッグ・レジスタを正しく設定しなければなりません．このレジスタに0を書き込むことで通常の動作をさせることができます．プログラムのデバッグ中には，適宜適切な値を設定します．

（7）Causeレジスタの設定

割り込みベクタに専用のアドレスを与えるため，CauseレジスタのIVビットに1を設定します．Cause[IV]が1のとき，割り込み発生時プロセッサはKSEG0 0x80000200，またはKSEG1 0xBFC00400（Status[BEV]でどちらかが選択される）からフェッチを開始します．

Cause[IV]が0の場合，割り込みはほかのエクセプションといっしょにひとまとめにされ，Cause[ExcCode]フィールドをデコードして調べなければなりません．したがって，Cause[IV]を1にするほうが，割り込みサービスの実行に早く着手できることになり，応答時間を短くできます．

（8）命令/データ・キャッシュの初期化

Au1xxxのキャッシュは，リセット後すぐに利用可能な状態に初期化されています．しかし，ラインタイム・リセット後などの場合を考えて，リセット前のデータや命令をキャッシュから取ってこないようにするためにエントリをすべてインバリデートしておくことを勧めします．

（9）TLBの初期化

Au1xxxのTLBは，リセット後自動的にはインバリデートされません．したがって，ブート・コードによって必ずインバリデートしなければなりません．偶然，エントリとして有効な値がヒットして，ランダムなアドレスに変換されたりすると，原因がわかりにくいバグになります．

（10）CPUコア動作クロックの設定

sys_cpupllレジスタに適切な値を設定することにより，Au1xxx CPUコアの動作周波数を決めます．メモリ・コントローラはCPU_PLLを利用するので，この設定は初期化の早い時期に行うことが必要です．Au1 CPUコアの動作周波数は，リセット後にはデフォルトで192MHzになります．

（11）システム・バス動作クロックの設定

システム・バスの動作周波数は，CPUコアのクロックを分周して利用しており，sys_powerctrlレジスタによって分周値を決めます．メモリ・コントローラや多くの内蔵ペリフェラルはシステム・バスのクロックを利用するので，この設定は初期化の早期に行うことが必要です．システム・バスの動作周波数は，リセット後デフォルトでCPUコアの動作周波数の1/2になります．

（12）AUX_PLLの設定

AUX_PLLは，内蔵ペリフェラル用のクロックです．sys_auxpllレジスタに適切な値を設定することでAUX_PLLの動作周波数を決めます．AUX_PLLもCPU_PLL同様，設定を変更するとPLLがロックし直すまで時間がかかります．

（13）32KHzクロックを起動

TOY/RTCが利用する32KHzのクロックを起動します．システムがTOY/RTCを利用する前に，32KHzクロックを起動しておかなければなりません．

（14）スタティック・バス・コントローラの初期化

ブートROMや，そのほかのメモリのサイズやメモリ・マップを正しく設定するために，スタティック・メモリ・コントローラを初期化しなければなりません．スタティック・メモリ・コントローラは，どのタイプのリセットによってもリセット前の値を保持しないため，必ず初期化し直さなければなりません．一方，SDRAMは，スリープ・モードのためセルフ・リフレッシュ・モードになっているかもしれないので，リセットの種類を調べた後で初期化します．

Au1xxxを用いたシステムでは，通常はスタティック・バス・コントローラに接続されたROM，またはフラッシュ・メモリからシステムをブートします．次に示す例は，より一般的な，Au1xxxをスタティック・バス・コントローラに接続されたROM/フラッシュ・メモリからブートする手順です．

Au1xxxプロセッサではリセット後，物理アドレス0x1FC00000に対してスタティック・バス・コントローラのチップ・セレクト $\overline{RCE0}$ がアサートされるよう，スタティック・バス・アドレス設定レジスタmem_staddr0が自動的に初期化されます．具体的には，mem_staddr0[E]が0b1に，mem_staddr0[CSBA]が0b00011111110000に，そしてmem_staddr0[CSMASK]が0b11111111111111になります．つまり，mem_staddr0[CSBA]の値によって，ベース・アドレスが0x1FC00000に，mem_staddr0[CSMASK]の値によって，アドレス範囲が256Kバイトに設定されます．これにより，リセット後，$\overline{RCE0}$ は，0x1FC00000～0x1FC3FFFFの256Kバイトのアドレス範囲に対してアサートされるよう，自動的に設定されます．

実際に使用しているROM/フラッシュ・メモリがこのサイズであれば，後はタイミング・パラメータをスタティック・タイミング・レジスタmem_sttime0に設定するだけです．mem_sttime0の各パラメータは，デフォルトですべて最大値が入るので，アクセス・タイムは非常に遅くなっています．

使用するROM/フラッシュ・メモリが256Kバイトではない場合，そのサイズが4Mバイトを越えるかどうかで，スタティッ

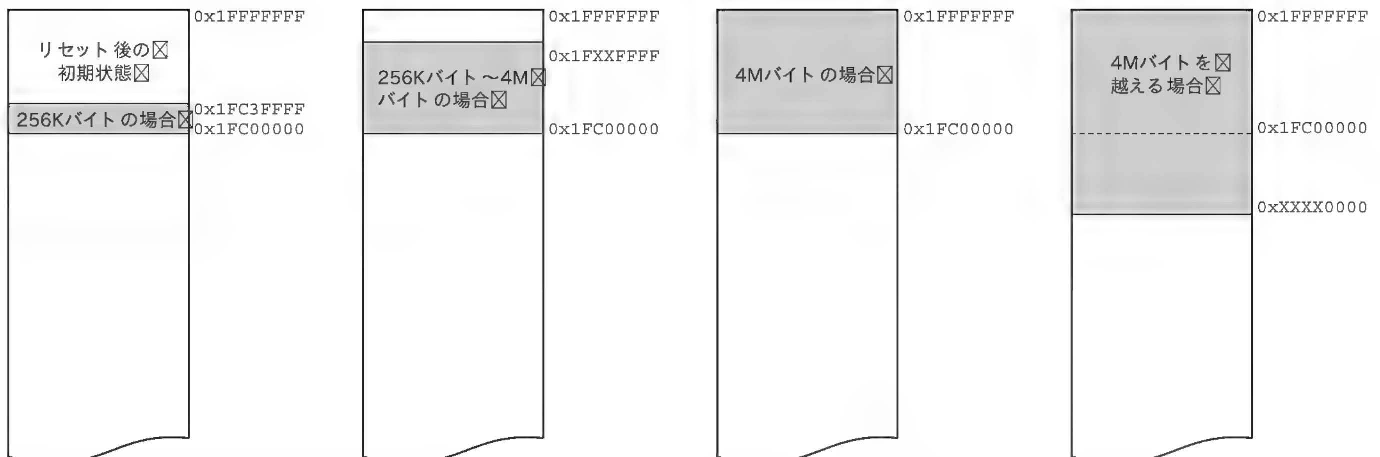


図21　ブートROM/フラッシュ・メモリの配置

表3　ブートROM/フラッシュ・メモリ用のmem_staddr0の設定値

ブートROMサイズ(バイト)	CSBAビット	CSMASKビット	ブートROMの物理ベース・アドレス	mem_staddr0
256K以下	0b00 0111 1111 0000	0b11 1111 1111 1111	0x1FC0 0000	0x11FC3FFF
512K	0b00 0111 1111 0000	0b11 1111 1111 1110	0x1FC0 0000	0x11FC3FFE
1M	0b00 0111 1111 0000	0b11 1111 1111 1100	0x1FC0 0000	0x11FC3FFC
2M	0b00 0111 1111 0000	0b11 1111 1111 1000	0x1FC0 0000	0x11FC3FF8
4M	0b00 0111 1111 0000	0b11 1111 1111 0000	0x1FC0 0000	0x11FC3FF0
8M	0b00 0111 1110 0000	0b11 1111 1110 0000	0x1F80 0000	0x11F83FE0
16M	0b00 0111 1100 0000	0b11 1111 1100 0000	0x1F00 0000	0x11F03FC0
32M	0b00 0111 1000 0000	0b11 1111 1000 0000	0x1E00 0000	0x11E03F80
64M	0b00 0111 0000 0000	0b11 1111 0000 0000	0x1C00 0000	0x11C03F00
128M	0b00 0110 0000 0000	0b11 1110 0000 0000	0x1800 0000	0x11803E00

ク・バス・アドレス設定レジスタmem_staddr0の設定方法が変わります(図21)．具体的な設定値を，表3に示します．

(15)内蔵ペリフェラルの初期化

リセット後，Au1xxxに内蔵されているペリフェラルを，既知の状態に初期化しなければなりません．また，使用しないペリフェラルはディセーブルすることにより，消費電力を少なくすることができます．初期化が必要なペリフェラル類は，次に示すとおりです．

- Frequency Generators(sys_freqctrl0, sys_freqctrl1, sys_clksrc)
- GPIO(sys_pininputen)
- AC97(ac97_enable)
- USBホスト(usbh_enable)
- USBデバイス(usbd_enable)
- IrDA(ir_enable)
- UARTs(uart_enable)
- MACs(macen_enable)
- I^2S(i2s_enable)
- SSIs(ssi_enable)

(16)リセットの原因を調べる

リセットの種類は，sys_wakesrcレジスタに記録されます．場合によっては，リセットのタイプによってブート・コードは，電源ONリセット(ハードウェア・リセット)，Sleepからの復帰，ランタイム・リセット，およびAu1550ではHibernateからの復帰の，それぞれに対応したパスに分岐する必要があります．

(17)SDRAMコントローラを設定

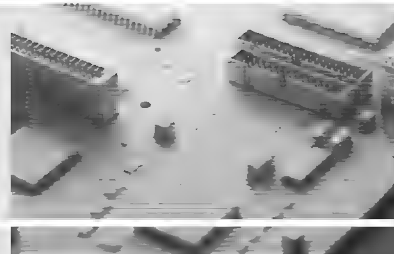
mem_sdmodeレジスタに，SDRAMの動作モードやアクセス・タイミング・パラメータなどを設定します．mem_sdaddrレジスタに，先頭アドレスmem_sdaddr[CBBA]とレンジmem_sdaddr[CSMASK]などを設定します．mem_sdconfiga, mem_sdconfigbレジスタで，リフレッシュのタイミングとクロックのイネーブルそのほかを設定します．

いとう・しん　日本AMD(株)

第2部 Chapter 5

超小型コンピュータTeacubeにも搭載されている

V_Rシリーズの概要とV_R5701の詳細



根木 勝彦/武田 憲一/小関 達也

本章では，まずV_Rシリーズの構成を解説し，V_Rシリーズの中でもハイエンドな情報家電をターゲットにしたV_R5701について詳しく解説する．最後にV_R5701を搭載した超小型コンピュータTeacubeを取り上げ，実際のプログラミング事例を解説する．

（編集部）

1 日本で設計・開発したV_Rシリーズ

● V_Rシリーズの開発コンセプト

V_RシリーズはNECエレクトロニクスが開発・製造を行っている，MIPS命令セットに準拠したRISC型マイクロプロセッサ群の総称です．

NECは1989年からMIPSプロセッサの開発を開始しました．当初約10年間の開発コンセプトは「シリコンバレーの最先端RISCプロセッサ設計技術と，日本の最先端半導体製造技術を融合させ，世界最高性能のプロセッサを創る」というものでした．その開発コンセプトの中でV_R4300やV_R10000など多くのヒット商品が生まれました．

この開発コンセプトは，時代のニーズに呼応して2000年ごろを境に大きく変化しました．ユビキタス・コンピューティングの時代といわれる中，プロセッサを求める人々の期待は「プロセッサという半導体そのもの」から「プロセッサを用いたソリューションやアイデア」へと変化しました．そのため，実際にプロセッサを設計したエンジニア自身が「自分のプロセッサの特徴や設計思想を熱く語れる」ことがたいへん重要になりました．NECのV_Rチームは90年代の終わりごろからこのことに着目し，（意外と知られていないが）現在ではすべての組み込み用途向けプロセッサを日本で設計・開発しています（図1）．

▶低消費電力・高性能組み込みプロセッサ

汎用性を維持し高性能を追及	V_R4131，V_R4133など
周辺回路をワンチップに集積	V_R4181，V_R4181Aなど

▶超高性能組み込みプロセッサ

パソコンの性能を情報家電に	V_R5500，V_R5701など
ハイエンド組み込みシステム向け	V_R7701，V_R9721など

▶高並列マルチプロセッサ対応

スーパ・コンピュータ用途	V_R14000，V_R16000など

図1 V_Rシリーズ・プロセッサのラインナップ

● V_R4100ファミリ――V_Rシリーズのエントリ・ファミリ

V_R4100ファミリは，低消費電力・省スペースを追求した高性能64ビットRISCプロセッサです．このシリーズはほかのV_Rシリーズに先駆け，90年代中頃からNECで独自開発を進めてきました．当初は情報携帯端末（PDA）とマイクロソフトのWindows CEサポートを強く意識した商品構成になっていました．むろん今でもその基本路線は踏襲しており，多くのWindows CE対応のPDAに採用されています．しかし現在では，対応するOSもLinux，NetBSD，VxWorks，ITRON，T-Kernelなどへと広がりをみせ，ブロードバンド・ルータ，ディジタル・カメラ（DSC），カーナビ，無線LANカード，各種業務用機器など，さまざまな分野へ急速に拡大しています．

現在のV_R4100ファミリは大きく分けて2系統あります．汎用性を維持しつつ高性能を追及するV_R413x系と，メモリ以外の必要な周辺回路をほとんど1チップに集積したV_R418x系です．

▶ 汎用性を維持し高性能を追及するV_R413x系

V_R4131は低電力で高性能な64ビットの4130CPUコアに，SDRAMメモリ・コントローラとPCIコントローラを集積したチップ構成になっています．4130CPUコアは，2ウェイ・スーパ・スカラのRISCプロセッサ・コアで，200MHz動作時に約340MIPSの性能を発揮します．V_R4131の消費電力は200MHz動作時でも220mW程度ときわめて少なく，多彩な省電力モードのサポートと相まって電池寿命の向上に貢献しています．また，熱の発生が少ないため省スペース化も可能となり，SDRAMやフラッシュ・メモリを搭載したMCM（マルチ・チップ・モジュール）なども製品化されています．V_R4131にEthernetコントローラ（2チャネル）や暗号処理機能を追加したV_R4133も発売されており，無線LANルータなどのネットワーク機器への搭載が進んでいます．

▶ 周辺回路を1チップに集積したV_R418x系

V_R4181は多くの周辺回路を1チップに集積したプロセッサで

COLUMN

VRチームの活動

VRシリーズのもう一つの特徴は，NECエレクトロニクスの開発・販売チームがパートナ各社と協力し，プロセッサのみならずCPUモジュールやボード・コンピュータまでをシームレスに開発・提供していることです（写真A）．それらの活動の一部として，表AにVRシリーズを搭載したボード・コンピュータをまとめました．活動の意義などは参考文献（1）などでも詳しく解説しているので，興味のある方はご覧ください．



写真A　VRシリーズ搭載ボード・コンピュータ群

表A　VRシリーズ搭載ボード・コンピュータのいろいろ

TB0193（L-Card+のハードウェア）	VR4181
TB0225（BGAタイプのMCM）	VR4131
TB0226（TB0225を用いたコンピュータ）	VR4131
TB0247（μT-Engineのハードウェア）	VR4131
VR4131DIMM（DIMMタイプのCPUモジュール）	VR4131
VR7701PMC（PMC基板タイプのCPUモジュール）	VR7701

▶（株）タンバック（http://www.tanbac.co.jp/）

VR5500-ATOM（超小型コンピュータ）	VR5500
VR5500-TE（T-Engineのハードウェア）	VR5500
SEMC5701（Teacubeのハードウェア）	VR5701

▶シマフジ電機（株）（http://www.shimafuji.co.jp/）

M-CARD（カード型Linuxコンピュータ）	VR4181A
M-Serverシリーズ（Linuxサーバ）	VR7701など

▶メガソリューション（株）（http://www.megasolution.jp/）

L-Card+（名刺サイズLinuxサーバ）	VR4181
L-CardA（名刺サイズLinuxサーバ）	VR4181A
L-Board（ワンボードLinuxサーバ）	VR4122

▶レーザーファイブ（株）（http://www.laser5.co.jp/）

N-Card（小型Linuxボード）	VR4131

▶（有）ハンブルソフト（http://www.humblesoft.com/）

μT-Engine/VR4131開発キット	VR4131
T-Engine/VR5500開発キット	VR5500
Teacube/VR5701評価キット	VR5701

▶パーソナルメディア（株）（http://www.personal-media.co.jp/）

す．CPU部分の動作周波数は66MHzとVRシリーズの中では低速ですが，周辺回路を含めた消費電力が約115mWと極小であることが特徴です．A-Dコンバータ，D-Aコンバータ，CompactFlashインターフェース，ISA系バス，GPIO，LCDコントローラ，キーボード・コントローラ，メモリ・コントローラなど，多くの周辺回路を集積しているのも特徴です．

VR4181の姉妹品としてVR4181Aがあります．CPUの1次キャッシュ・メモリの容量が2倍になり，動作周波数が131MHzまで引き上げられました．周辺機能としてはUSB1.1のホストとファンクション・コントローラ，AC97，IDEインターフェース，I²Cなどが追加されています．VR4181とVR4181Aは世代交代して単純に置き換わるものではなく，用途によって使い分けられています．

● VR5500ファミリ――低消費電力の64ビット・プロセッサ

VR5500はレーザ・プリンタ，セット・トップ・ボックス（STB），HDDレコーダなどのディジタル家電，ネットワーク機器，カーナビ，ロボット制御など，さまざまな分野に応用できる消費電力2Wクラスの高性能64ビット・プロセッサです．浮動小数点演算ユニット（FPU）も搭載しています．

おもな特徴は次の三つです．

- 従来のMIPSプロセッサからの継承性を重視
- 1GHzが狙えるアーキテクチャ設計
- 純国産プロセッサであること

▶ 従来のMIPSプロセッサからの継承性を重視

VR5500は，数多くの採用実績がある従来のMIPSプロセッサからの継承性を重視して開発されました．MIPSプロセッサのCPUバスはSysADと呼ばれるものですが，各プロセッサによって若干の違い（方言）があり，そのまま置き換えるのが困難でした．VR5500はVR5432やVR5000はもちろん，VRシリーズ以外の同クラスのMIPSプロセッサのバス・モードも一部サポートしています．したがって，プロセッサに接続されるASICやチップセットを変更することなく，VR5500に置き換えて性能向上を図ることができます．また，MIPSプロセッサには32ビット幅のSysADと，64ビット幅のSysADがあります．VR5500はモード切り替えにより，その両方に対応できるよう設計されています．

▶ 1GHzが狙えるアーキテクチャ設計

今まで，MIPSプロセッサは「MHzあたりの性能は良いが動作周波数が高くない」と言われていました．この点については

いろいろな意見や議論がありますが，やはりプロセッサは高い動作周波数を追求しなければならない運命にあります．一般にパイプラインの段数を増やせば高い動作周波数を達成するのが容易になりますが，MHzあたりの性能は低下します．ここがマイクロアーキテクチャ設計の腕の見せどころです．

V_R5500は10段パイプラインです．MHzあたりの性能低下を補完するため，アウト・オブ・オーダ命令処理，強化された分岐予測機構，投機実行，デカップルド・アーキテクチャにより最適化されたリソース（2命令発行，6個の実行ユニット，3命令完了）などを採用し，V_R5432（5段パイプライン）と同等のMHzあたりの性能を達成しています．

これらの機能により400MHz動作時に約800MIPSの性能を達成しています．ちなみにアウト・オブ・オーダとは，命令の依存関係を調べて実行できる命令から実行するプロセッサの性能を向上させる技術です．現在多くのMIPSベンダから提供されているMIPSプロセッサの中で，アウト・オブ・オーダ処理ができるのは，後述するV_R10000ファミリとV_R5500だけです．

V_R5500はNECエレクトロニクスの150nmプロセスを使用し，400MHz動作品が量産中です．800MIPSの性能により，MPEG系の動画データをCPUで直接デコードすることが可能となり，インターネット・アプライアンス装置の心臓部として多く使われています．

▶ 純国産プロセッサ

V_R5500は，企画・設計・開発・生産をすべてNECが単独で行った純国産プロセッサです．V_R5500を使用する装置開発者がV_R5500を設計・開発したエンジニアと直接対話できることの意味は重要です．お互いに多くのことを学べるからです．その意味でも，このV_R5500コアは標準プロセッサのみならず，ASSPやASCPなどのシステムLSIへも採用が進んでいます．

このV_R5500をCPUコアとして内蔵し，メモリ・コントローラやPCI，IDEなどの周辺回路を集積したのがV_R5701です．このプロセッサは次節で詳しく解説します．

● V_R7700ファミリ――V_R5500コア製品

前述のV_R5500をCPUコアとして，2次キャッシュ・メモリ，SDRAMインターフェース，I/Oインターフェースなどを集積したのがV_R7700ファミリです．

その第一弾の製品がV_R7701で，256Kバイトの2次キャッシュ・メモリ，64ビット/133MHzのDDR-SDRAMインターフェース，64ビット/133MHzのPCI-Xインターフェース，100M/10M対応Ethernetコントローラ（2チャネル）などを集積しています．V_R7701は現在量産出荷中です．このプロセッサはネットワーク機器（ルータ），ディスク・ストレージ装置（RAID），サーバなどのインターネット基幹系の機器など，低消費電力で高いデータ処理性能が要求される装置を設計・開発している技術者が注目しているプロセッサです．

V_R7701は400MHz（約800MIPS）ですが，CPUコアを800MHz（約1600MIPS）に高速化し，2次キャッシュを512Kバイトに増やし，Ethernetコントローラをギガビットに対応させたV_R9721の開発も進み，現在は装置メーカと共同で評価が進んでいます．V_R7700ファミリはコンピュータとしての基本機能を1チップに集積しているため，メモリを接続するだけで強力なコンピュータ装置を安価に作ることが可能です．今後が楽しみなファミリです．

● V_R10000ファミリ――マルチプロセッサを構成可能

組み込み用途ではありませんが，V_Rシリーズにはもう一つのファミリがあります．NECとシリコングラフィックス社（以下SGI）が共同開発をしているV_R10000ファミリです．このシリーズは最大16Mバイトまでの大容量2次キャッシュを128ビット幅で接続できるインターフェースや，強力な浮動小数点演算ユニット（FPU）などをもつ4並列の64ビットRISCプロセッサです．マルチプロセッサを構成するための機構を備えています．

▶ スーパ・コンピュータ向けの超高性能プロセッサ

V_R10000ファミリがもっとも使われているのは，SGIのOriginシリーズというスーパ・コンピュータです（http://www.sgi.co.jp/origin/3000/overview.html）．このスカラ型高並列スーパ・コンピュータは，米国のロスアラモス国立研究所（6144プロセッサ）をはじめとして世界中で稼動しており，核物理シミュレーション，核融合プラズマの研究開発，ゲノム研究，気象・天候の解析などに貢献しています．

▶ 性能あたりの消費電力を極力抑える

スカラ型高並列スーパ・コンピュータの世界では，プロセッサ1個の性能もさることながら，限られた体積の筐体の中にどれだけ多くの性能（＝どれだけ多くのプロセッサ）を詰め込めるかが勝負となります．したがって，プロセッサの消費電力がたいへん重要な意味をもちます．

たとえば128個のプロセッサを筐体に収めるとき，プロセッサ1個の消費電力が70Wの場合，プロセッサだけで10kW近くの熱が発生します．NECでは早い時期からこのことに注目し，消費電力を抑えて性能を向上させることを目標としてきました．現在出荷中のV_R16000/800MHzが，その規模や性能の割には消費電力が20W以下と少ないのはこのためです．この高性能・低消費電力が高く評価され，V_R10000ファミリは基幹系組み込みシステムの世界でも活躍しています．〈根木 勝彦〉

2 周辺回路集積型プロセッサ V_R5701

V_R5701は，高性能なCPUパワーを手軽に利用できるよう開発された周辺回路集積型のプロセッサです．CPUコアにはNECエレクトロニクスが独自に設計したアウト・オブ・オーダ実行が可能な2ウェイ・スーパ・スカラ方式の64ビットCPU（MIPS Ⅳアーキテクチャ準拠）を採用しています．

このV_R5701により高度な情報化社会に要求される複雑な処理を低コストで実現することが可能になります．

表1　V_R5701のおもな仕様

CPUコア	MIPS IV準拠 V_R5500　64ビット RISC CPU@336MHz
メモリ・インターフェース	DDR-SDRAM@133/100MHz
外部バス・インターフェース	32ビット PCI(Rev2.1準拠)@33MHz
	16ビット・ローカル・バス
内蔵コントローラ	USB1.1ホスト，AC97，UltraDMA(モード4)IDE，DMA(4チャネル)，
	GPIO(最大43チャネル，割り込み32チャネル)，32ビット・タイマ(4チャネル)
	同期・非同期シリアル(CSI/UART)，JTAG
電源	CPUコア 1.6V，DDR-SDRAM 2.5V，その他IO 3.3V
パッケージ	352ピン ABGA(35mm²，26×26 4Row，1.27mmピッチ)

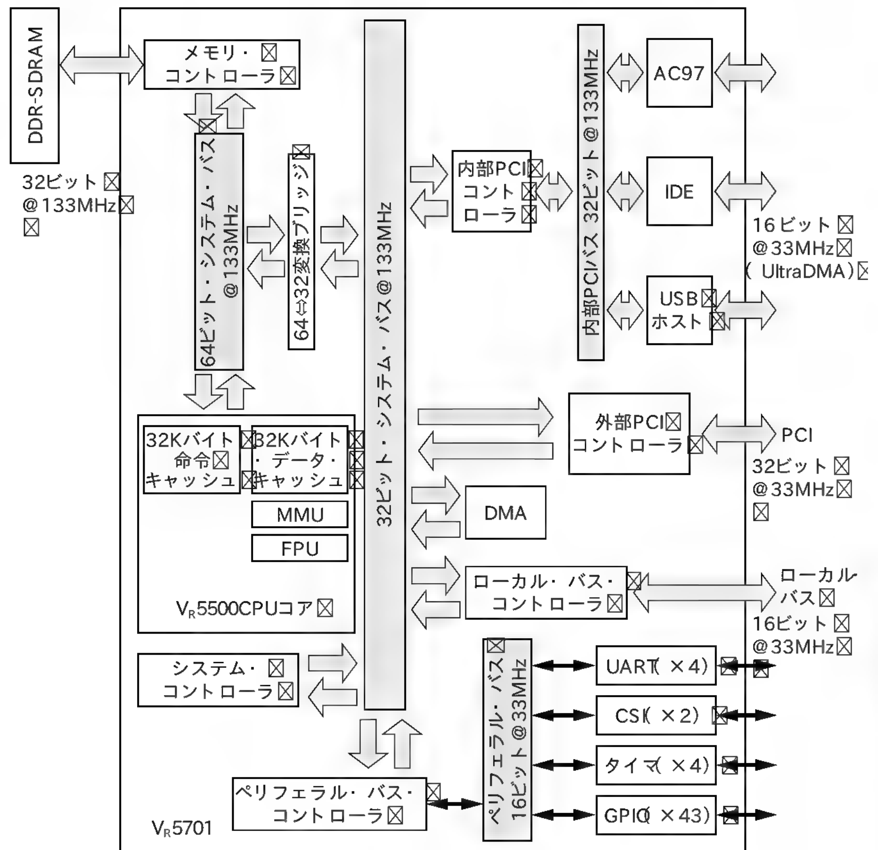


図2　V_R5701のブロック図

● 超小型コンピュータ SEMC5701に採用されたV_R5701の概要

表1にV_R5701のおもな仕様を，図2にブロック図を示します．メモリにはDDR-SDRAMインターフェースを採用しており，32ビット/133MHz(データ・バスはダブル・レート)でメモリを直結することが可能です．内蔵のCPUコアとメモリ・コントローラ間は，高速な転送レートを生かすため64ビット/133MHzの内部バスで結んでいます．内部システム・バスは32ビット/133MHzで動作し，応答が遅いデバイスがバスを開放する間に別のリクエストを処理できる遅延トランザクションを実装しています．このため，速度の異なるさまざまなデバイスが接続されている内部システム・バスを効率的に利用できます．また，アクセスが集中するメモリ・コントローラに対しては，CPUからのリクエストとは別に内部システム・バス側から二つのアクセスを同時に受け付けることが可能な構造になっており，バス・トランザクションを極力減らしてメモリ・アクセス性能を向上させる設計となっています．

外部バスにはPCI Rev.2.1準拠32ビット/33MHzのPCIバスと，33MHzの16ビット・ローカル・バスがあります．ローカル・バスはおもに起動用のROMを接続する用途に使われます．データ帯域確保のため外部PCIバスとは別に内部デバイス用にもPCIコントローラが搭載されており，AC97コントローラ，UltraDMA対応IDEコントローラ，USB1.1ホスト・コントローラが接続されています．そのほかの周辺機能としては割り込みコントローラ，DMAコントローラ(4チャネル)，同期/非同期シリアル・コントローラ，32ビット・インターバル・タイマ(4チャネル)，GPIO(43本，ただし割り込み32本，一部兼用端子)を備えています．

V_R5701は外部用にPCIバス・インターフェースを備えており，今日幅広く流通しているPCIデバイスが接続可能です．PCIバスはシステム・バスとは非同期で動作しているので，PCIバスの動作周波数を自由に設定できます．また，ATA/ATAPI-5準拠のIDEコントローラを内蔵しているので，ATA/ATAPI-5準拠のデバイスと最大転送レート66.6Mバイト/秒(UltraDMAモード4)でデータ転送が可能で，高速なHDDとの接続も可能です．大容量化・高速化の進むCompactFlashをTrueIDEモードで利用することによりソリッド・ステート・システムを容易に構築することもできます．またUSB1.1コントローラにより，キーボードやマウスなどの豊富な外部デバイスを容易に接続することができます．

このような構成のV_R5701と豊富にそろうPCIデバイスを接続することにより，V_R5701の強力なCPUパワーを利用したシステム開発を容易に行えます．次にV_R5701の核となるV_R5500 CPUコアについて説明します．

● アウト・オブ・オーダ実行可能なV_R5500CPUコア

V_R5701は，NECエレクトロニクスによる独自開発のV_R5500 CPUコアを採用しており，その特徴は次のようなものです．

- MIPS IV準拠の64ビットCPU
- 2ウェイ・スーパ・スカラ方式スーパ・パイプライン
- アウト・オブ・オーダ実行

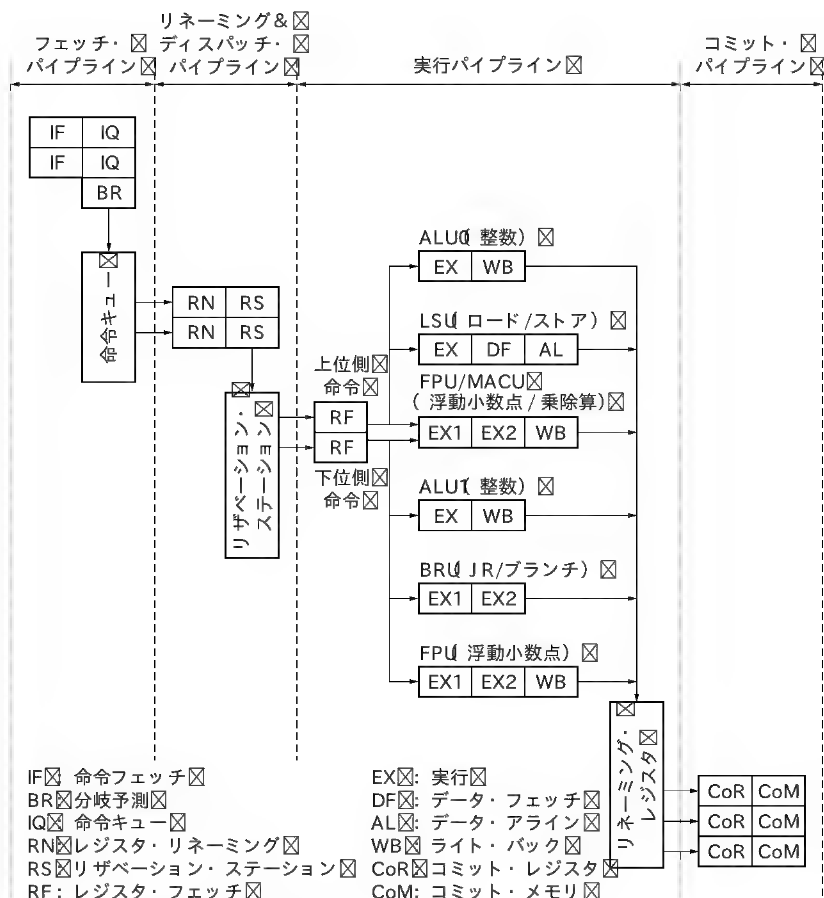


図3　V_R5500コアのパイプライン構成

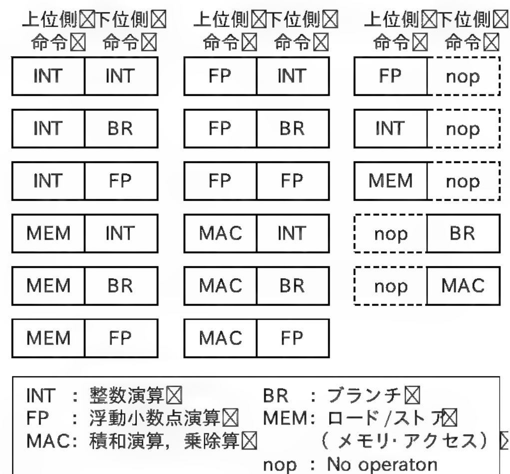


図4　同時実行可能な命令

●TLB搭載（48エントリ）による仮想アドレス・サポート
●4Kバイトの分岐履歴テーブルによる分岐予測機構
●命令/データそれぞれ32Kバイトのキャッシュ（2ウェイ）
●浮動小数点演算ユニット
●整数積和演算などマルチメディア演算機能のサポート

ユーザ・モードでの64ビット命令の実行許可は，CPUのモード設定により切り替え可能で，従来の32ビット命令も実行可能です．OSで利用するカーネル・モードではつねに64ビット命令が使用できますが，64ビット命令を使用する場合には割り込み時のレジスタの退避を64ビット命令で行います．

次に図3にパイプラインのブロック図を示します．命令の実行は次のように行われます．

1）命令フェッチ

命令のフェッチと分岐予測に基づく実行フローの制御を行います．

2）リネーミング＆ディスパッチ

命令の依存関係を調べレジスタ・リネーミングにより依存関係を解消します．レジスタ・リネーミングでは解消できない依存関係に基づいて，依存関係のない順（アウト・オブ・オーダ）に命令を発行します．

3）実行

二つの命令が同時に実行可能です．実行可能な命令の組み合わせを図4に示します．

4）コミット

アウト・オブ・オーダで実行された命令によるレジスタの更新を実命令順に更新します．また例外/トラップのチェックも行われます．

●レジスタ・リネーミングとアウト・オブ・オーダ実行

図3に示すように，深いパイプライン構造の場合，命令の依存関係によるパイプライン・ストールが問題となります．また，スーパ・スカラにより命令の同時実行が可能であるため，実行パイプラインに効率よく命令を供給していく必要があります．この問題を解決するためV_R5500コアでは，レジスタ・リネーミングとアウト・オブ・オーダ処理を行います．

図5にアウト・オブ・オーダ実行可能な構成時の効果について示します．ここでは説明を簡単にするため，命令の同時実行はされないものとします．図5（a）に示すようにイン・オーダで実行している場合，命令に依存関係があるとパイプラインがストールします．このとき，仮のレジスタを割り当てる（レジスタ・リネーミング）ことでパイプラインのストールを回避することができます．たとえば次のような場合，

```
addu  r2,  r1,  r2    # r2=r1+r2
sw    r3,  0x0(r2)    # *r2=r3
                    ↑ アドレス計算のため r2 の計算を待つ
li    r3,  0x020      # r3=0x020
addu  r3,  r3,  r4    # r3=r3+r4
```

の2行目でアドレス計算にr2の値が必要なので，1行目の計算が終わるまで命令が実行できず，パイプライン・ストールが起こります．3行目のliはレジスタr4が上書きされてしまうため，swが終わるまで実行することができません．しかし，ここ

で仮のレジスタ，たとえばt3を動的に割り当てることが可能であれば依存関係を解消することができます．

```
addu  r2,  r1,     r2
sw    r3,  0x0(r2)
li    t3,  0x020
addu  t3,  t3,     r4
```

これにより，sw命令とli命令の実行順序が変わっても，プログラムの実行結果は保証されます．したがって，sw命令がr2の計算結果を待っている間にli命令を発行するよう命令を入れ換えることが可能です．

```
addu  r2,  r1,     r2
li    t3,  0x020
sw    r3,  0x0(r2)
addiu t3,  t3,     r4
```

このようにレジスタ・リネーミングを行って命令間の依存関係を解消することにより，ディスパッチ・ユニットがリザベーション・ステーションからより多くの依存関係のない命令を探してきて，アウト・オブ・オーダで実行することが可能になります．また，仮のレジスタを割り当てることで，分岐履歴に基づく投機実行を行って予測がまちがっていた場合でも，実レジスタへの値の更新は行われていないため，命令の取り消しを行うことができるようになります．

アウト・オブ・オーダで実行した順にそのままレジスタを更新してしまうと，PC(プログラム・カウンタ)の指す命令より先の命令によりレジスタが更新された状態が発生してしまい，割り込みがかかったときなどにレジスタ内容の一貫性が取れなくなってしまいます．そこでコミットメント・ステージにより，先に実行された命令によるレジスタ更新を保留し，実命令順にレジスタの更新を行うようイン・オーダで命令を完了させ，PCとレジスタの一貫性を保つようにしています．このようにアウト・オブ・オーダで実行した状態を図5(b)に示します．

アウト・オブ・オーダ実行では図5(b)の命令3が完了したときに示されるように，命令完了が同時に3個行われている箇所があります．これは，命令依存関係が解消された場合はアウト・オブ・オーダ実行された命令がいくつも完了するのを待っている状態になり，この溜まった命令を掃き出すために同時に完了できる命令数を多く取っているためです．この数はV_R5701では3個となっています．

● V_R5500コアとコードの最適化

なぜコンパイラで命令実行順序を最適化できるのにハードウェアで命令実行順序を変える必要があるのでしょうか．図5(a)に示すようにパイプライン・ストールが起きないように入れ替える必要がある命令数は，パイプラインの深さに依存します．特定のパイプライン数に依存して最適化してしまうと，CPUのアーキテクチャが変わってパイプラインの段数が変わるたびにバイナリを作り直す必要があります．速度を重要視してハードウェアを意識したコードをアセンブラで書くことも可能

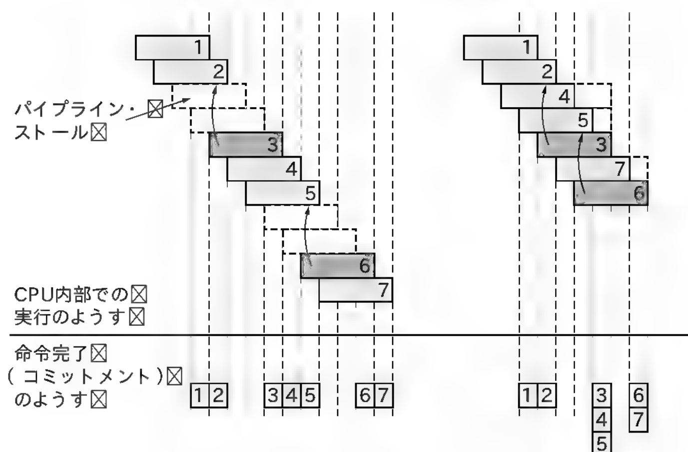


図5　アウト・オブ・オーダ実行例

ですが，パイプライン段数を固定した最適化は，CPUの世代が変わると再度最適化が必要になるといった理由により，現実的ではありません．製品ラインナップでローエンド，ハイエンドの差別化をする場合，CPUの世代が違うことは十分にありえる話です．この問題を解決する手段としてスーパ・スカラの能力を十分に発揮できるアウト・オブ・オーダ実行機能がたいへん有効になります．

もちろんV_R5500コアでも，大量のデータ処理時にキャッシュ・ミスが予想される場合に，データのプリフェッチを行ったり，図4に示す同時実行可能な命令の組み合わせになるようコードを最適化すれば，ピーク性能を絞りだすことも可能です．しかしながら，たとえコードが最適化されていない状態でも，ハードウェアで極力効率を上げる機構をもつV_R5500コアは，実際の使用時においてたいへん強力です．

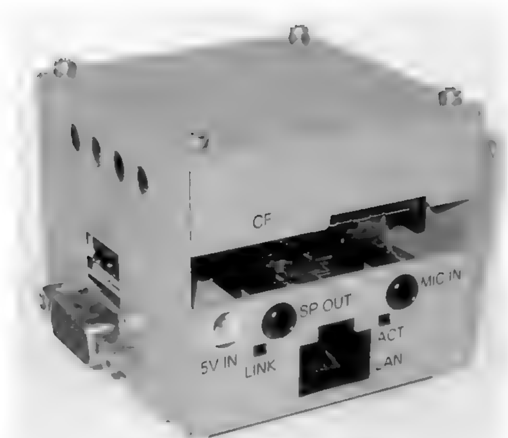
ちなみに現在，このV_R5500コアを用いた製品としてはV_R5500(単体CPU)，V_R5701，V_R7701(256Kバイトの2次キャッシュを搭載し，64ビットの強力なI/Oをもつハイエンド・プロセッサ)，μPD61160(ディジタル・ハイビジョン・テレビ用システムLSI)などがあります．〈武田 憲一〉

3 V_R5701を使った超小型コンピュータの活用事例

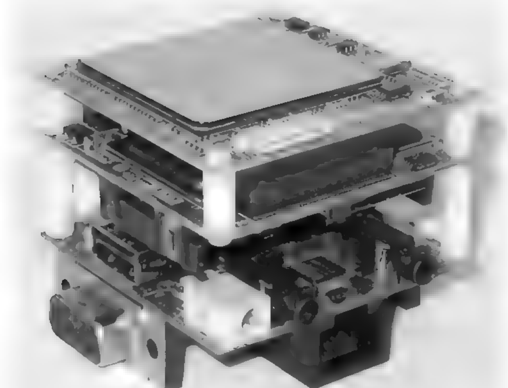
● 超小型コンピュータSEMC5701の仕様

次はV_R5701を内蔵した超小型コンピュータSEMC5701(写真1)を使用し，OSを介さずにMIPSプロセッサを直接制御するプログラムをいくつか実行させてみたいと思います．

図6にSEMC5701のブロック図，表2に仕様の一覧を示します．SEMC5701には最大SXGAのアナログRGB，USB2.0(2チャネル)，Ethernet，ステレオ音声入出力(AC97)など，一般的なパソコンが備える機能が，約5cmの小さな立方体に凝縮されています．デジカメなどの普及により，低価格化が進む



（a）ケースに入れた状態



（b）ケースを外した状態

写真1　超小型コンピュータ SEMC5701の外観

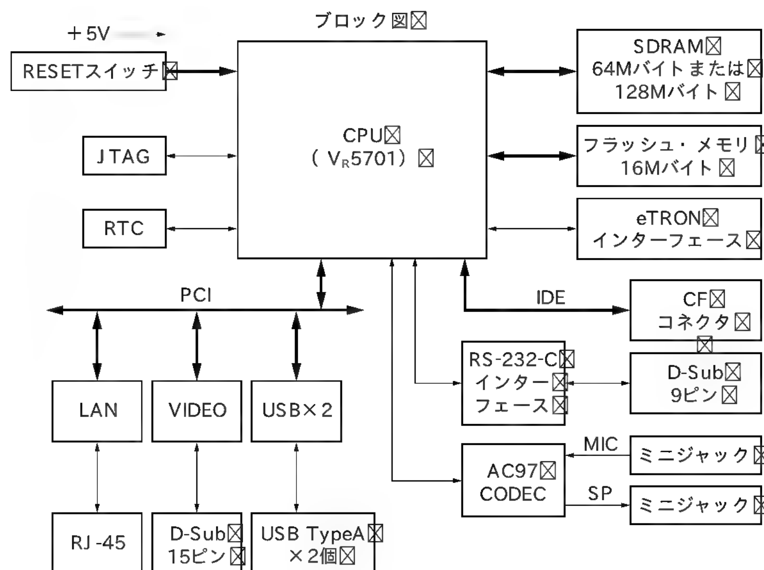


図6　超小型コンピュータ SEMC5701のブロック図

表2　超小型コンピュータ SEMC5701の仕様

CPU	V_R5701　266M/333MHz
フラッシュ・メモリ	16Mバイト
RAM	DDR-SDRAM 64M/128Mバイト
入出力インターフェース	RTC，eTRONインターフェース，CompactFlash（TrueIDE），SXGA（1280×1024），LAN（10M/100M），USB2.0×2，オーディオ（ステレオ）入出力，RS-232-C，JTAGインターフェース（デバッグ用）
電源	+5V
外形寸法	52×52×45mm（突起部除く）

CompactFlashカードをTrueIDEモードで接続してHDDの代わりにしています．小さいことはもちろんですが，低消費電力（CPU待機時で約3.5W，333MHz動作時で約6W）が特徴で，ファンレスで無音化を実現しています．

SEMC5701は，PMON（後述）というモニタが搭載された形でシマフジ電機から購入可能です．なお，このハードウェアにT-KernelやT-ShellなどのTRON系OSをポーティングした形で，パーソナルメディア社から売り出されているのが「Teacube/V_R5701評価キット」です．

● 開発環境のセットアップ

今回，開発プラットホームには，Linuxを走らせたPCを使用しました．筆者が使用したコンパイラは，`http://source.rfc822.org/pub/mirror/ftp.ds2.pg.gda.pl/pub/macro/RPMS/i386/`からダウンロードしたもので，ファイル名は`mipsel-linux-binutils-2.13.2.1-2.i386.rpm`と`mipsel-linux-gcc-2.95.4-8.i386.rpm`です．rpmなので，次のコマンドでPCにインストールできます．

```
# rpm -ivh mipsel-linux-binutils-2.13.2.1-2.i386.rpm
# rpm -ivh mipsel-linux-gcc-2.95.4-8.i386.rpm
```

これでコンパイラのインストールができたはずです．gccはコンパイル時のオプション指定とリンク・ディレクティブ・ファイルを作成するのが少しめんどうです．コンパイル・オプションは次のようにしました．

```
# mipsel-linux-gcc -mips4 -EL -mno-abicalls
  -fno-pic -mlong-calls -nostdlib
  -fno-builtin -Wl, -Map, phoe.map -Tphoe.ld
  led.c -o led
```

上の例の`led.c`や`led`はソース・コードのファイル名とバイナリ・ファイル名です．別のプログラムの場合は適宜変更してください．リンク・ディレクティブ・ファイルをリスト1に示します．

● PMONについて

シマフジ電機からSEMC5701を購入すると，サンプルとしてフリー・ソフトウェアのPMONというモニタ・プログラムがオンボードのフラッシュ・メモリに書き込まれています．作成したプログラムをターゲットにロードする際にはICEを使用す

リスト1　リンク・ディレクティブ・ファイル

```
SECTIONS
{
     _xfer = 1 ;
     _stack_init = 0x81010000 ;
     _heep_size = 0x20000 ;
     .text 0x81000000 :
     {
          _stext = . ;_rom_base = . ;
          *(.text)
          *(.gnu.linkonce.t*)
          . = ALIGN(4) ;
          _ctor_list = . ;
          *(.ctors);*(.ctor);LONG (0);
          _dtor_list = . ;
          *(.dtors);*(.dtor);LONG (0);
          _etext  = . ;
     }
     .rodata :
     {
          *(.rodata)
          *(.gnu.linkonce.r*)
          _erdata  = . ;
     }
     .data :
     {
          _data  = . ;
          *(.data)
          *(.gnu.linkonce.d*)
     }
     _gp = . + 0x8000;
     .sdata :
     {
          *(.lit8)
          *(.lit4)
          *(.sdata)
          _edata  = . ;
     }
     .sbss : {
          _fbss = . ;
          *(.sbss)
          *(.scommon)
     }
     .bss : {
          *(.bss)
          *(COMMON)
          _end = . ;
          _heep = . ;
     }

     .debug          0 : { *(.debug) }
     .debug_srcinfo  0 : { *(.debug_srcinfo) }
     .debug_aranges  0 : { *(.debug_aranges) }
     .debug_pubnames 0 : { *(.debug_pubnames) }
     .debug_sfnames  0 : { *(.debug_sfnames) }
     .line           0 : { *(.line) }

     .debug_info     0 : { *(.debug_info) }
     .debug_abbrev   0 : { *(.debug_abbrev) }
     .debug_line     0 : { *(.debug_line) }
     .debug_frame    0 : { *(.debug_frame) }
     .debug_str      0 : { *(.debug_str) }
     .debug_loc      0 : { *(.debug_loc) }
     .debug_macinfo  0 : { *(.debug_macinfo) }
}
```

リスト2　PMON起動時のコンソールの表示のようす

```
PCI slot 26/0: NEC USB (serialbus, USB)
PCI slot 26/1: NEC USB (serialbus, USB)
PCI slot 26/2: NEC, product: 0xe0 (serialbus, USB)
PCI slot 27/0: Intel, product: 0x1209 (network, ethernet)
PCI slot 28/0: vendor/product: 0x126f/0x0720 (display, VGA)
fx0: Intel i82559ER Ethernet, rev 9
fx0: Ethernet address 01:01:01:01:01:01, 10/100 Mb/s

PMON version 1.4.902 [EL,FP,NET]
NEC Electronics Inc. Dec 24 2003 15:26:33
This is free software, and comes with ABSOLUTELY NO WARRANTY,
you are welcome to redistribute it without restriction.
CPU type VR5500.  Rev 2.0.  266.65 MHz/133.33 MHz.
Memory size  64 MB.
Icache size  32 KB, 32/line (2 way)
Dcache size  32 KB, 32/line (2 way)

NEC01>
```

るのが一般的ですが，今回はモニタ・プログラム(PMON)を使ってターゲットにロードして実行してみましょう．

リスト2は，PMONがSEMC5701に組み込まれている状態で電源を投入したときの，シリアルUART(デフォルトでは9600bps)に出力されるログです．“NEC01>”の部分はPMONのプロンプトです．ここからさまざまなPMONのコマンドを入力することができます．

今回使用するのはtftpのプログラムのロード機能と，ロードしたプログラムの実行機能です．プログラムをtftpでロードするには，たとえば，

```
NEC01>boot  192.168.0.8: led
```

と入力します．ここで192.168.0.8はtftpサーバのIPアドレス，ledはロードするファイル名です．IPアドレス/ファイル名は適宜変更してください．

プログラムを実行するには，

```
NEC01> g -e 81000000
```

と入力します．ここで，8100000はプログラムのエントリ・アドレスです．

ここまででプログラムを実行させる環境が整いました．それ

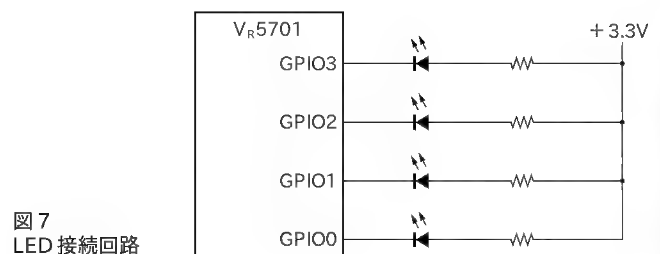
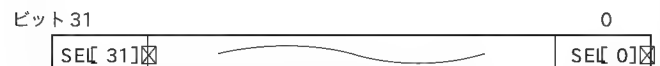


図7
LED接続回路

ビット31　0

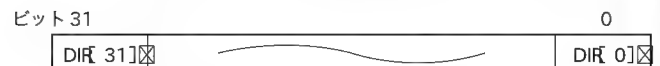


すべてリード/ライト可，リセット値：0xffff ffff

ビット[31:0]　SEL　GPIO[31:0]機能セレクタ
1=GPIO端子(リセット値)
0=兼用端子

(a) GIU_FUNSEL0レジスタ(オフセット：0x0960)

ビット31　0



すべてリード/ライト可，リセット値：0x0000 0000

ビット[31:0]　DIR　GPIOの方向
1=出力
0=入力(リセット値)

(b) GIU_DIR0レジスタ(オフセット：0x0950)

ビット31　0



すべてリード/ライト可，リセット値：0x0000 0000

GIU_PIO0
ビット[31:0]　PIOD　GPIOポート入出力データ
1=“H”レベル
0=“L”レベル(リセット値)

(c) GIU_PIO0レジスタ(オフセット：0x0940)

図8　LED点灯用GPIO制御レジスタ

では実際にプログラムを動作させてみましょう．

● LEDプログラム(led.c)

SEMC5701には，ユーザ設定用のLEDが四つ付いています．このプログラムはこれらのLEDを点灯させるものです．図7はLEDの接続図，図8はプログラムで操作するV_R5701のLED点

リスト 3　LED 点灯制御プログラム

```
asm("
        .text
        .align  2
        .globl  _start
        .ent    _start
_start:
        la      $29, _stack_init
        la      $28, _gp

        la      $8, main
        jr      $8
        nop
        nop
        .end    _start
");


int main(int argc,char **argv)
{
        unsigned long *VR5701_GPIO_func
                      =(unsigned long *)0xbe000960 ;
        unsigned long *VR5701_GPIO_dir
                      =(unsigned long *)0xbe000950 ;
        unsigned long *VR5701_GPIO_data
                      =(unsigned long *)0xbe000940 ;
        * VR5701_GPIO_func |=  0x0f ;
        * VR5701_GPIO_dir  |=  0x0f ;
        * VR5701_GPIO_data &= ~0x0f ;
        while(1) ;
        return 0;
}
```

リスト 4　メモリ/レジスタ・ダンプ・プログラムの実行結果(dump.c)

```
NEC01> boot 192.168.0.8:dump
Loading file: 192.168.0.8:dump (elf)
0x81000000/2832 + 28 syms
Entry address is 81000000
NEC01> g -e 81000000
Input Addr -> be000000
BE000000 | 1E00008F 00000000 00000000 00000000
BE000010 | 00000000 00000000 00000000 00000000
BE000020 | 00000000 00000000 00000000 00000000
BE000030 | 00000000 00000000 00000000 00000000
BE000040 | 000000AA 00000000 00000080 00000000
BE000050 | 00000000 00000000 00000000 00000000
BE000060 | 00000000 00000000 00000000 00000000
BE000070 | 00000000 00000000 00000000 00000000
BE000080 | 1F00004C 00000000 00000000 00000000
BE000090 | 00000000 00000000 00000000 00000000
BE0000A0 | 00000000 00000000 00000000 00000000
BE0000B0 | 00000000 00000000 00000000 00000000
BE0000C0 | 1000008C 00000000 1800008D 00000000
BE0000D0 | 00000000 00000000 00000000 00000000
BE0000E0 | 1880008D 00000000 1900008D 00000000
BE0000F0 | 00000000 00000000 00000000 00000000
Input Addr ->
```

表 3　dump.c の関数

int strlen(char *ss)	文字列 ss の長さを返す
void strcpy(char *s1, char *s2)	s1 に s2 をコピーする
char *LtoH(long longY,char *p)	long 型変数 Y を文字列(p)に変換する
char *LtoH0(long longY, char *p)	LtoH と同じだが，文字列長が 8 バイト未満のときは左に 0 を付け 8 バイトの文字列にする
int HexToLong (char *p, unsigned long *lv)	文字列を long 型変数に変換する
void init_uart()	シルアル UART を初期設定する
void puts(char* p)	シルアル UART に文字列を出力する
void gets(char* p)	シルアル UART から文字列を取得する
void printh32(unsigned long dd)	シルアルＵＡＲＴに long 型変数を文字列で出力する

表 4　V_R5701 のベース・アドレス・レジスタのオフセット

XXX0_0000h	内部レジスタ
XXX0_0040h	メイン・メモリ(バンク 01)
XXX0_0048h	メイン・メモリ(バンク 23)
XXX0_0080h	ローカル・バス CS0
XXX0_0088h	ローカル・バス CS1
XXX0_0090h	ローカル・バス CS2
XXX0_0098h	ローカル・バス CS3
XXX0_00C0h	外部 PCI ウィンドウ 0
XXX0_00C8h	外部 PCI ウィンドウ 1
XXX0_00E0h	内部 PCI ウィンドウ 0
XXX0_00E8h	内部 PCI ウィンドウ 1

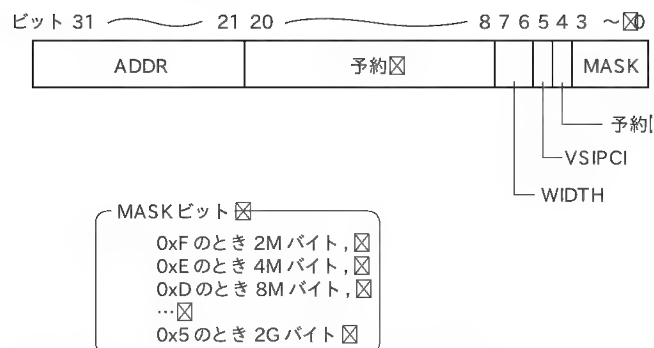


図 9　V_R5701 のベース・アドレス・レジスタの構造

灯用 GPIO 制御レジスタです．まず端子をポート機能に設定し，ポートのディレクションを出力に，最後にデータ("H"/"L")をレジスタに書き込みます．これで LED が点灯します．

プログラム・リストをリスト 3に示します．前述したコンパイル指定でコンパイルし，バイナリを tftp サーバのディレクトリにコピーして，PMON のプロンプトから boot コマンドと g コマンドを実行します．

● メモリ/レジスタ・ダンプ・プログラム(dump.c)

2番目のプログラムはメモリ/レジスタ・ダンプ・プログラムです．プログラムで作成した関数を表 3に示します．プログラムの処理はシリアル UART を初期化し，ユーザからアドレスとなる数値を入力，そのメモリ内容を表示するものです．このプログラムで V_R5701 のレジスタをのぞいてみることにしましょう．

LED プログラムと同様にロードした dump プログラムで，BE00_0000h 番地を表示させたものがリスト 4で，V_R5701 の各空間のベース・アドレス・レジスタの内容が表示されています．

V_R5701 は，メイン・メモリやローカル・バスの CSn をマップするアドレスを任意に設定できる構造になっています．図 9に V_R5701 のベース・アドレス・レジスタの構造を，表 4に各ベース・アドレス・レジスタのオフセットを示します．

リスト 4を見ると，オフセット 0000h(内部レジスタのベー

領域	アドレス
フラッシュ・メモリ	1F00_0000h
	1F00_0000h
	1E1F_FFFFh
内部レジスタ	
	1E00_0000h
	197F_FFFFh
内部PCIウィンドウ1	
	1900_0000h
内部PCIウィンドウ0	
	1880_0000h
外部PCIウィンドウ1	
	1800_0000h
	10FF_FFFFh
外部PCIウィンドウ0	
	1000_0000h
	003F_FFFFh
SDRAM	
	0000_0000h

図10 SEMC5701/PMONのメモリ・マップ

ス・アドレス・レジスタ)の値は`1E00_008Fh`となっているので，内部レジスタは物理アドレス`1E00_0000h`から2Mバイトの領域にマップされていることになります．`dump.c`の結果からPMONの物理アドレスのメモリ・マップを作成すると図10のようになります．

V_R5701のベース・アドレス・レジスタ群は，当然ながら内部レジスタ空間にマッピングされるので，内部レジスタのベース・アドレス・レジスタのアドレス設定を書き換えると，書き換えた瞬間に新しいアドレス空間にマップし直されるので注意が必要です．ちなみに，このレジスタのリセット直後のデフォルトの設定は`1FA0_008Fh`です．

● 音声の録音再生プログラム(audio.c)

最後のプログラムは音声の録音再生プログラムです．V_R5701にはAC97が内蔵されています．SEMC5701の場合，V_R5701のAC97インターフェースにはAC97CODECデバイスWM9707CFT/V(WOLFSON製)が接続されています．準備としてはSEMC5701にマイクとスピーカを接続します．

プログラムはマイクから音声を入力させ，SDRAMに音声データを保存し，その音声データを再生するものです．プログラムのフローを図11，プログラムの実行のようすをリスト5に示します．

まとめ

SEMC5701はパソコン並の入出力インターフェースを備えており，高性能なMIPSプロセッサを搭載しているという，非常にユニークな超小型コンピュータです．今回は比較的簡単なプログラムを紹介しました．これを機に，SEMC5701の大きく開かれたプログラミングの可能性にトライして見てはいかがでしょうか．

〈小関 達也〉

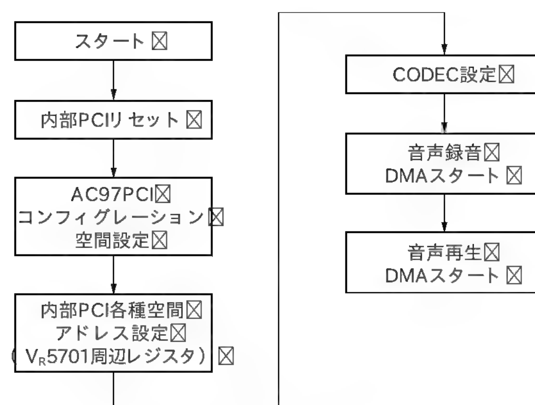


図11 音声の録音再生プログラムの動作フロー

リスト5 音声の録音再生プログラムの実行結果(audio.c)

```
NEC01> boot 192.168.0.8:audio
Loading file: 192.168.0.8:audio (elf)
0x81000000/4336 + 28 syms
Entry address is 81000000
NEC01> g -e 81000000
Start record !・・・・・・・・・ 録音開始
stop record ? (y/n) > y
Play Start ? (y/n) > y・・・・・ 再生開始
stop play ? (y/n) >
```

参考文献
(1) 特集「Webベースのハードウェア制御」，2002年8月号，トランジスタ技術，pp.143～230，CQ出版
(2) 組み込みシステム技術者向けオンライン・マガジンEIS，門田浩のユビキタスウォッチ，「第4回UCサロン」(http://www.caravan.net/eis/)
(3) 成松宏，第2特集「CPUモジュールを使用したLinuxワンボードコンピュータ開発記」，Embedded UNIX Vol.6，pp.85～121，CQ出版
(4) 桑原健，T-Engine(n101)/uT-Engine(n301M)のハードウェア(前編)，TRONWARE Vol.77，pp.10～20，パーソナルメディア
(5) 桑原健，T-Engine(n101)/uT-Engine(n301M)のハードウェア(後編)，TRONWARE Vol.78，pp.29～37，パーソナルメディア
(6) 山本満博，「マイコン単体ではなくボード/モジュール単位でLSI製品を提供する」，2003年2月号，Design Wave Magazine，pp.122～127，CQ出版
(7) V_Rシリーズパンフレット(http://www.necel.com/micro/product/vr/U15575JJ3V0PF00.pdfよりダウンロード可能)
(8) V_R5500ユーザーマニュアル，NECエレクトロニクス
(9) V_R5701ユーザーマニュアル，NECエレクトロニクス

ねき・かつひこ/たけだ・けんいち NECエレクトロニクス(株)
こせき・たつや シマフジ電機(株)

第2部 Chapter 6 32ビット・コアのTX19/39から64ビット・コアのTX49/99まで

TXシリーズとT-Engine/TX4956の概要



吉田 俊哉/寺尾 隆宏/黒瀬 浩史

本章ではTXシリーズ全体のシリーズ展開やロードマップを解説したあと，最新プロセッサであるTX4937とTX4938について，そしてTX4938を搭載したリファレンス・ボードRBHMA4500について解説する．そして最後にT-Engine/TX4956についての概要を解説する．

（編集部）

はじめに

TXシリーズ・マイクロプロセッサは，MIPSアーキテクチャに基づき東芝が独自に開発した"TXシリーズ・コア"を搭載した，32ビット/64ビットのRISCプロセッサです．ここではTXシリーズの概要と，TX4956Cを搭載したT-Engineボードについて解説します．

1 TXシリーズの構成

● TXシリーズのプロセッサ・コア・ロードマップ

図1にTXシリーズのプロセッサ・コアのロードマップを示します．TXシリーズは東芝のASICと同一のプロセスで開発されています．ローエンドのTX19シリーズから最上位のTX99シリーズまで，幅広い性能をもったコア・ラインナップがあります．

● おもなターゲット

図2にTXシリーズの主たるターゲット・マーケットを示します．汎用製品だけでなく，TXシリーズ・コアを使ったASSPも開発されています．ディジタル情報家電機器，ネットワーク機器，OA機器など幅広い分野で使われています．

● CPUコアの特徴

図3に各CPUコアの特徴を示します．低消費電力なコアから高性能コアまで，シームレスなアーキテクチャを提供しています．

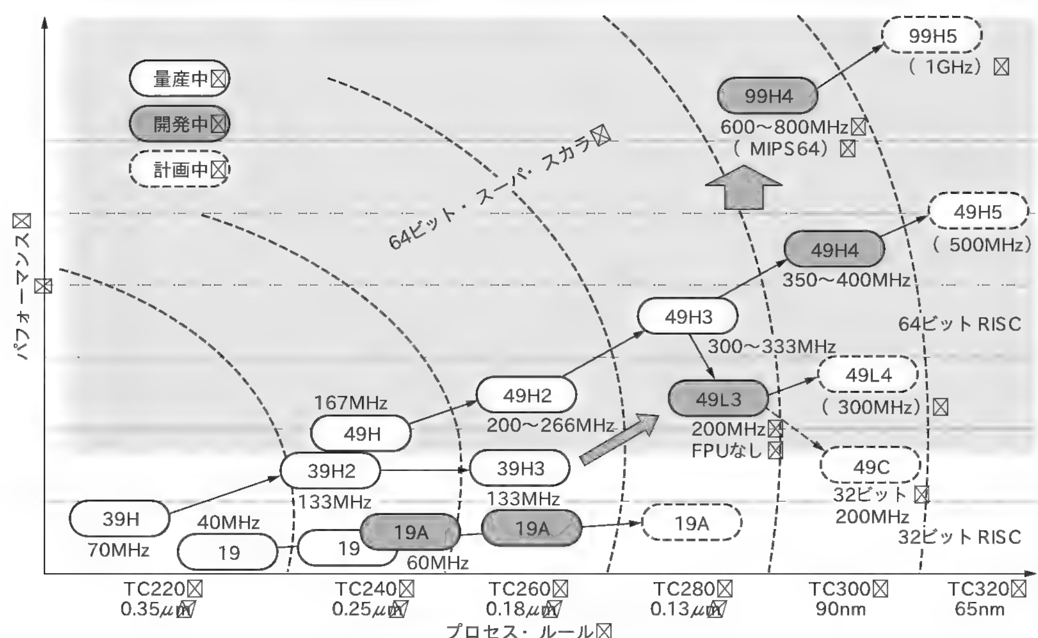


図1 TXシリーズのプロセッサ・コアのロードマップ

● TXシリーズ汎用製品ラインナップ

TXシリーズは現在，TX19，TX39，TX49の各シリーズで汎用製品を販売しています．表1～表3に各シリーズの製品機能一覧を示します．

2 最新のCPU TX4937/4938の特徴

ここでは，最新のCPU製品であるTX4937とTX4938の2製品について特徴などを説明します．

● TX4937の構成と機能

TX4937はおもに，ネットワーク機器やOA機器，ディジタル情報家電機器で使われています．図4にTX4937のブロック図を示します．おもな特徴は次のとおりです．

- 動作周波数：300MHz，および333MHz
- TX49/H3コア：
  内蔵キャッシュ・メモリ：命令32Kバイト/データ32Kバイト
  浮動小数点演算ユニット(単精度/倍精度)
  EJTAGデバッグ・サポート・ユニット
- おもな内蔵周辺機能
  メモリ・コントローラ(SDRAM，NOR型フラッシュ・メモリ，SRAM)
  PCIバス・コントローラ(66MHz)

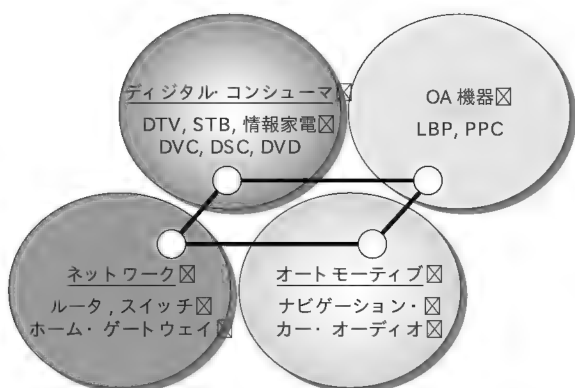


図2 TXシリーズのおもなターゲット

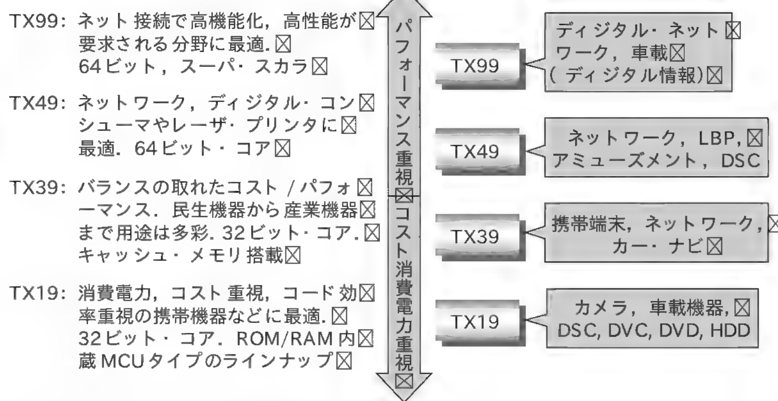


図3 TXシリーズ・コアの特徴

表1 TX39シリーズ製品機能一覧

型名	TMPR3911BU	TMPR3911BXB	TMPR3912AU-92	TMPR3912XB-92	TMPR3916F	TMPR3927CF
最大動作周波数(MHz)	58	58	92	92	60	133
外部データ・バス幅(ビット)	32	32	32	32	32	32
内部データ・バス幅(ビット)	32	32	32	32	32	32
命令キャッシュ容量(バイト)	4K	4K	4K	4K	4K	8K
データ・キャッシュ容量(バイト)	1K	1K	1K	1K	1K	4K
内蔵ROM	なし	なし	なし	なし	なし	なし
内蔵RAM	なし	なし	なし	なし	なし	なし
DMAコントローラ(チャネル)	—	—	—	—	2	4
メモリ・コントローラ	SDRAM，SRAM，ROM，フラッシュ・メモリ	SDRAM，SRAM，ROM，フラッシュ・メモリ	SDRAM，SRAM，ROM，フラッシュ・メモリ	SDRAM，SRAM，ROM，フラッシュ・メモリ	SDRAM，DRAM，SRAM，ROM，フラッシュ・メモリ	SDRAM，SRAM，ROM，フラッシュ・メモリ
シリアル・インターフェース(チャネル)	3	3	3	3	4	2
I/Oポート数(本数)	39	39	39	39	30	16
タイマ(チャネル)	2	2	2	2	1	3
外部割り込み(本数)	39	39	39	39	3	6
PCMCIAインターフェース(チャネル)	2	2	2	2	—	—
LCDコントローラ	○(STN)	○(STN)	○(STN)	○(STN)	○(TFT)	—
CANコントローラ(チャネル)	—	—	—	—	2	—
PCIコントローラ	—	—	—	—	—	Rev. 2.1 33MHz，外部バス・マスタ4本
アナログ・フロントエンド	TC35143	TC35143	TC35143	TC35143	—	—
パッケージ	LQFP176	FBGA177	LQFP208	FBGA217	QFP208	QFP240
その他	MMU, RTC	MMU, RTC	MMU, RTC	MMU, RTC	DSU	DSU, MMU

表2　TX49シリーズ製品機能一覧（プロセッサ・タイプ）

型　名	TMPR4955AF-200	TMPR4955BFG-300	TMPR4955CFG-400	TMPR4956CXBG-400
最大動作周波数（MHz）	200	300	400	400
外部データ・バス幅（ビット）	32	32	32	64
内部データ・バス幅（ビット）	64	64	64	64
命令キャッシュ容量（バイト）	32K	32K	32K	32K
データ・キャッシュ容量（バイト）	32K	32K	32K	32K
内蔵ROM	―	―	―	―
内蔵RAM	―	―	―	―
DMAコントローラ	―	―	―	―
メモリ・コントローラ	―	―	―	―
10/100Base-T EtherMAC	―	―	―	―
シリアル・インターフェース	―	―	―	―
I/Oポート数（本数）	―	―	―	―
タイマ（チャネル）	1	1	1	1
外部割り込み（本数）	7	7	7	7
PCMCIAインターフェース（チャネル）	―	―	―	―
PCIコントローラ	―	―	―	―
パッケージ	QFP160	QFP160	QFP256	PFBGA217

表3　TX49シリーズ製品機能一覧（周辺コントローラ内蔵タイプ）

型　名	TMPR4925XB-200	TMPR4926XB-200	TMPR4927ATB-200	TMPR4937XB-300	TMPR4938XB-300
最大動作周波数（MHz）	200	200	200	300	300
外部データ・バス幅（ビット）	32	32	64	64	64
内部データ・バス幅（ビット）	64	64	64	64	64
命令キャッシュ容量（バイト）	16K	16K	32K	32K	32K
データ・キャッシュ容量（バイト）	16K	16K	32K	32K	32K
内蔵ROM	―	―	―	―	―
内蔵RAM	―	―	―	―	―
DMAコントローラ	4	4	4	4	4
メモリ・コントローラ	NAND型フラッシュ，SDRAM，SRAM，ROM，NOR型フラッシュ	NAND型フラッシュ，SDRAM，SRAM，ROM，NOR型フラッシュ	SDRAM，SRAM，ROM，フラッシュ・メモリ	SDRAM，SRAM，ROM，フラッシュ・メモリ	NAND型フラッシュ，SDRAM，SRAM，ROM，NOR型フラッシュ
10/100Base-T EtherMAC	―	―	―	―	2
シリアル・インターフェース	2	2	2	2	2
I/Oポート数（本数）	32	32	16	16	16
タイマ（チャネル）	3	3	3	3	3
外部割り込み（本数）	8	8	6	6	6
PCMCIAインターフェース（チャネル）	2	2	―	―	―
PCIコントローラ	Rev. 2.2 33MHz	Rev. 2.2 33MHz	Rev. 2.2 66MHz/33MHz	Rev. 2.2 66MHz/33MHz	Rev. 2.2 66MHz/33MHz
コンパニオン・チップ	TC86C001	TC86C001	TC86C001	TC86C001	TC86C001
パッケージ	PBGA256	PBGA256	TBGA420	PBGA484	PBGA484

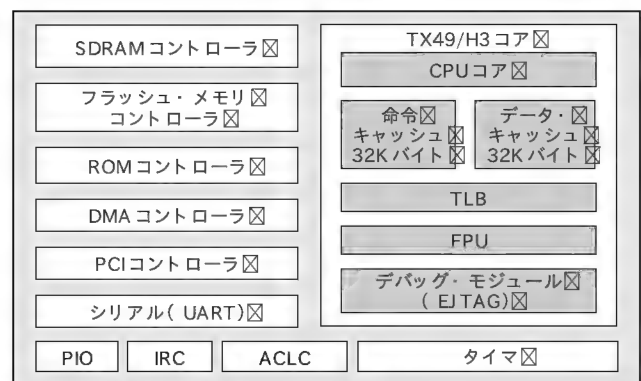


図4　TX4937ブロック図

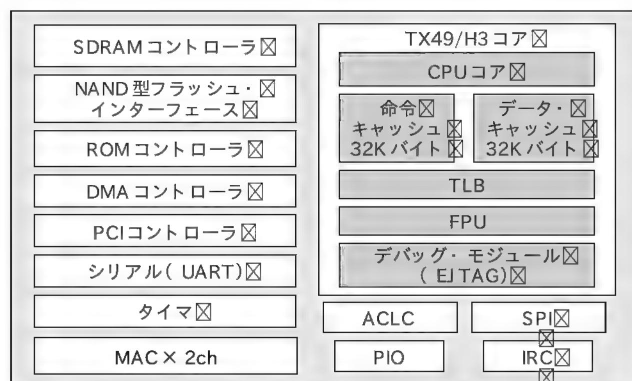


図5　TX4938ブロック図

- 動作電圧：内部1.5V，I/O 3.3V
- パッケージ：484ピン，TBGA

● TX4938 の構成と機能

TX4938は，10/100Base-T EtherMACを2チャネル，NANDフラッシュ・メモリを直結できるインターフェースを内蔵しています．そのため，特に高速なネットワーク・データ転送を必要とするブロードバンド関連機器や，大容量のプログラム/データ処理を必要とするディジタル情報家電機器に最適です．図5にTX4938のブロック図を示します．おもな特徴は次のとおりです．

- 動作周波数：300MHz，および333MHz
- TX49/H3コア：
  内蔵キャッシュ・メモリ：命令32バイト/データ32Kバイト
  浮動小数点演算ユニット（単精度/倍精度）
  EJTAGデバッグ・サポート・ユニット
- おもな内蔵周辺機能
  メモリ・コントローラ（SDRAM，NOR型フラッシュ・メモリ，SRAM）
  NAND型フラッシュ・メモリ・インターフェース回路
  10/100Base-T EtherMAC 2チャネル
  PCIバス・コントローラ（66MHz）
- 動作電圧：内部1.5V，I/O 3.3V
- パッケージ：484ピン，TBGA

〈吉田 俊哉〉

3 TX シリーズ対応リファレンス・ボードの構成とソフトウェア

● リファレンス・ボードの概要

東芝は，TXSystemRISCを簡単に使用できるように，各種

表4 RBHMA4500の仕様

項目		内容
CPU	TX4938	TMPR4938XB-300 動作周波数：CPUコア 300.0 MHz （G-BUS 100.0 MHz，水晶発振器 25.0 MHz）
メモリ	NOR型フラッシュ・メモリ	SYSTEM ROM 16Mバイト，4M×16ビット×2，32ビット・バス BOOT ROM 4Mバイト，1M×16ビット×2，32ビット・バス
	NAND型フラッシュ・メモリ	32Mバイト，32M×8ビット，8ビット・バス
	SO-DIMM	128Mバイト，64ビット・バス（最大実装サイズ512Mバイト可能）
	EEPROM	1Kビット，128×8ビット（×3）
主要インターフェース	SIO	2チャネル（9ピンD-sub） CPU内蔵SIOを使用（常時使用可能は，1チャネル）
	内蔵Ethernet	2チャネル（8ピン・モジュラ・ジャック，100Base-TX/10Base-T） CPU内蔵MAC+外部PHYを使用 各チャネルにMIIピン・アサイン準拠40ピン・コネクタ併設
	ACリンク	ACリンク・コネクタ搭載 （ただし，±12V供給は，PCIカード・エッジ接続時のみ）
	IDE	40ピン・コネクタ（PIOモードのみサポート）
	RTC（SPI）	リアルタイム・クロックを使用（電池CR2032実装済み）
PCI		PCIカード・スロット：1スロット，3.3V仕様 PCIカード・エッジ：3.3V仕様
デバッグ関連インターフェース	デバッグ用Ethernet	1チャネル（8ピン・モジュラ・ジャック，10Base-T） 一体型10 Mbps Ethernetコントローラを使用
	EJTAG	38ピン/40ピン・コネクタ
	ROMエミュレータ	50ピン専用コネクタ×2
	LED	18個24LED（内ユーザ開放：8LED）
	テスト・ピン	41端子（実装8端子）
	DIPSW	53ビット（内ユーザ開放：8ビット）
汎用インターフェース	リセット/NMI	共用モーメンタリ・スイッチ搭載（テスト・ピン併設）
	割り込み	5チャネル（INT[4：3]はピン・マルチプレクスあり）
	DMA	8チャネル（内4チャネルは，AC-link使用済）
	タイマ	2チャネル
	PIO	16ビット（PIO[7]以外ピン・マルチプレクスあり）
	スタッキング・コネクタ	2コネクタ（裏面未実装）
外形寸法		256.5 mm×122.0 mm（突起部除く）
重量		約0.45 kg（予定）
電源		+5V±5% 電流2A（予定） 電源供給元はPCIカード・エッジ，または電源コネクタ （PCIカード・エッジの場合は，+12VをAC-linkが使用）
動作温度		温度：5～35℃
保存温度		温度：－20～60℃
付属品		DCケーブル，SIO変換専用ケーブル，スペーサ，CD-ROM

図6　RBHMA4500のブロック図

リファレンス・ボードを提供しています．現在販売中のTX19，TX39，TX49の各TXシリーズのCPUに対してリファレンス・ボードを販売しています（TX3927対応ボードについては，東芝情報システムより販売）．

リファレンス・ボードを提供しているおもな目的として，次の点を重視しています．

- できる限り安価であること
- 動作評価が容易であること
- 実機ボードを設計する上で参考にする価値がある資料が存在すること
- プロトタイプ・ソフトウェア開発への対応

今回は，TX4938に対応したリファレンス・ボードRBHMA4500について解説します．表4（p.117）にその仕様を，図6にブロック図を，写真1に外観を示します．

● TX4938対応リファレンス・ボードRBHMA4500

TX4938は，TX3927から始まってTX4927，TX4937と続いてきたPCIバス・コントローラを内蔵したCPUで，本ボードもPCIカードを接続し，機能を評価することに主眼を置いて設計されています．PCIは，ホスト機能だけではなくターゲット動作の評価も行えるように，PICMGに準拠したホスト・ボード仕様と，通常のPCIカードの機能を融合させた構成となっています．

RBHMA4500には，TX4938が内蔵する豊富な機能を評価するために必要な周辺回路のほとんどを，小型なPCIカード上に実装しています．5V単一電源で使用でき，主要機能としてTX4938が内蔵する10/100M Ethernet MACを使用したEthernetコントローラを2チャネル（以降，内蔵Ethernet），交換可能な32MバイトNAND型フラッシュ・メモリ（ソケット実装），各種のOS搭載を考慮した大容量128MバイトSO-DIMM，16MバイトNOR型フラッシュ・メモリ，加えてATA（IDE）インターフェース，ACリンク・インターフェース，SPI接続のEEPROM，RTCが搭載されています．

デバッグ環境としては，RS-232-Cシリアル・ポート，10Base-T Ethernetコントローラ，EJTAGコネクタ，ROMエミュレータ・コネクタなど，さまざまなデバッグ機器が接続可能です．

● 動作モード

PCIに関しては，PCIカード・エッジ・コネクタを1スロット装備しており，直接3.3V仕様のPCIカードを接続することができます（CPUの電圧トレラント仕様で3.3V PCI専用）．

別売りの3.3V仕様PCIバック・プレーンを使用すると，二つのモードが使用可能です．

▶ ホスト・モード

ホスト・モードはバックプレーン上のCPUスロットに接続して動作するモードです．4枚の5V/3.3V共用PCIカードを33MHz動作で接続できます．バックプレーンは，標準的なPCのフレーム（筐体）に組み込めます．

▶ サテライト・モード

サテライト・モードは通常の3.3V仕様PCIカードとして使用できます．組み込みシステムでは，サブCPUとして使用されることもあり，接続評価などに使用できます．

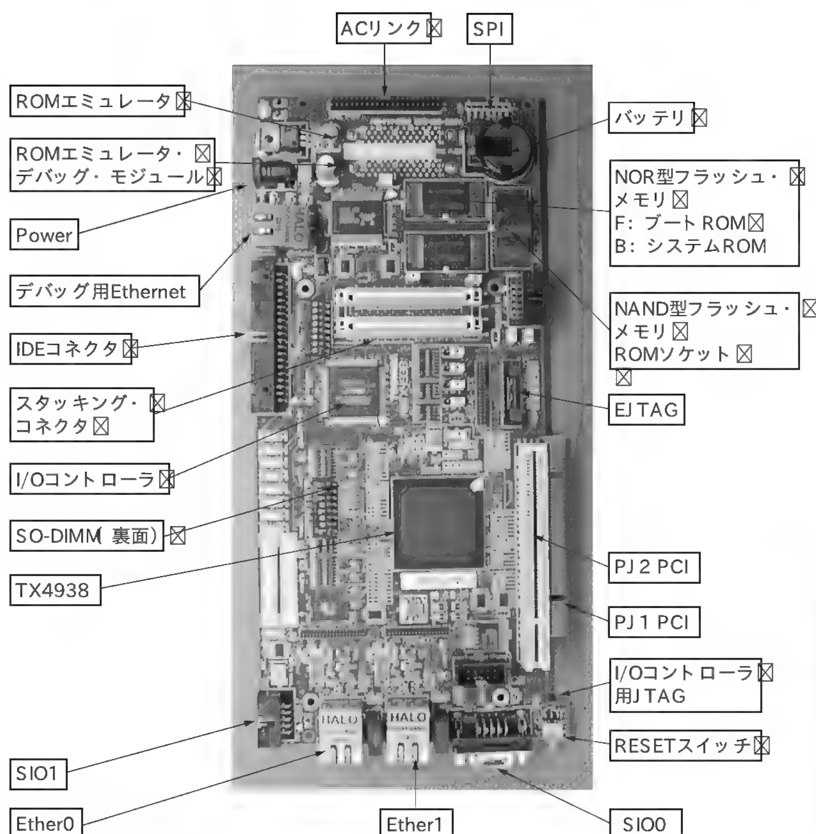


写真1 RBHMA4500の外観

非常に豊富な機能を限定されたピン数の中に収めているので，ファンクションは，ピン・マルチプレクスされています．本ボードでは，すべての機能が使用できるようになっており，動作設定は，ボード上のディップ・スイッチの設定で切り替えられます．

特に，100Base-TX Ethernet MACにつながるPHYは，リファレンスに採用したIC以外のICとの接続性を評価する場合，追加基板経由でPHYメーカ各社のMIIモジュールを接続可能にしています．

本ボードに搭載し切れなかった一部の回路は，拡張ボードの形で接続できるようになっています．それらは，TXシリーズで共通の仕様となっているので，TX4938以外のボードとの共用が可能です．

● ボード搭載ソフトウェア

RBHMA4500のブートROMには，ボード・モニタ・プログラムが実装されています．ボード・モニタ・プログラムには，標準のモニタとして，MIPS Technologies社が開発したROMモニタであるYAMONを採用し，サブ・モニタとして，RAMレスで動作する簡易ROMモニタをそなえています．

二つのモニタ・プログラムは，ディップ・スイッチの切り替えでブートするプログラムを選択できます．いずれも，コントロールにはPCを接続する必要があり，RBHMA4500のシリアル・ポートを使用しています．

これらモニタのソース・コードは，回路図データを含めて提供可能な体制をとっています．

● デバッグ環境

TX4938に内蔵されているEJTAG機能を使い，プログラムの動作をデバッグすることができます．

RBHMA4500においては，横河ディジタル・コンピュータ（株），京都マイクロコンピュータ（株），（株）コンピューテックス，ウインドリバー（株），（株）ソフィアシステムズの対応するEJTAGツールが使用できます．

ツール・メーカごとにEJTAGの接続方式が違うので，ジャンパ・ピンなどで設定できる仕様になっています．PCトレース機能に対応しているEJTAGでは，PCトレース・データを取得可能です．

GHS社製のマルチ・デバッガに対応したGMONもポーティングずみで，専用オブジェクト・コードが標準で添付されています．

● OS対応

RBHMA4500は，TX4938をサポートするOSの標準サポート環境になっており，標準サポートOSとなっているウインドリバーのVxWorks，MontaVista Linux，東芝情報システムのμITRONがポーティング済みです．

〈寺尾 隆宏〉

表5　TX4956の概要

コア		RISCコアTX49/H4搭載
パイプライン		5段パイプライン
オンチップ・キャッシュ 4ウェイ・セット・アソシアティブ・ロック機能	命令キャッシュ	32Kバイト内蔵
	データ・キャッシュ	32Kバイト内蔵
MMU・TLBエントリ		48double
システム・インターフェース		SysADバス(64/32ビット)
浮動小数点演算		単/倍精度FPU
消費電力		0.6W
電源電圧		内部：1.2V 外部：3.3Vまたは2.5V
デバッグ・ポート		DSU(EJTAG)
最大動作周波数		400MHz
パッケージ		217ピンPFBGA

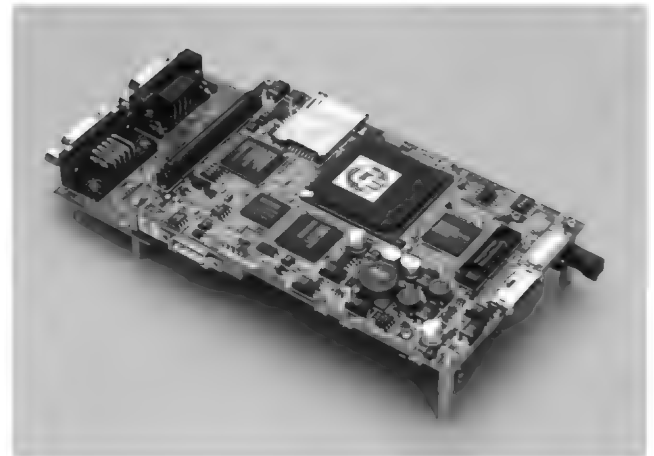


写真2　T-Engine/TX4956の外観

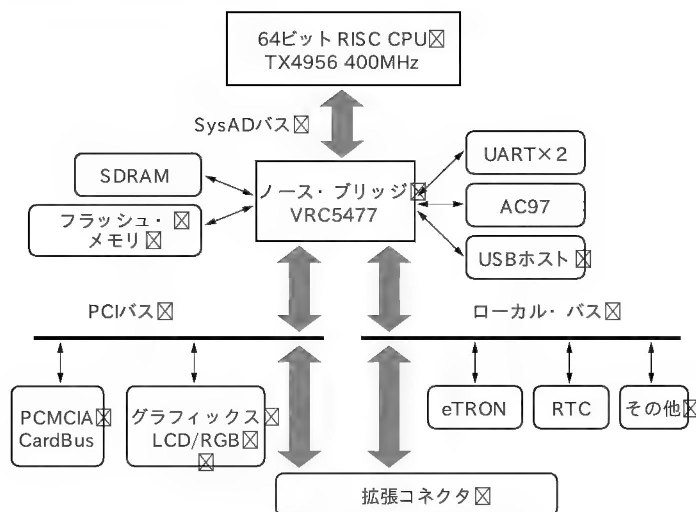


図7　T-Engine/TX4956のブロック図

4　T-Engine/TX4956の構成とソフトウェア

最後にTX4956を搭載したT-Engineボードの概要について解説します．なお本製品は，2004年4月に東芝情報システムよりリリースされています．東芝および東芝情報システムは，T-Engineツール・キットの開発を行いました．

● T-Engineとは

T-Engineとは，従来日本が得意としてきた携帯電話や情報家電に代表されるコンパクトで高機能な電子機器の開発を，来たるべきユビキタス・コンピューティング環境に向け標準化し，開発コストや開発期間を短縮するために考えられました．

組み込み機器を制御するためのソフトウェアの開発は，高機能化する機器と厳しいリソースの制約により困難をきわめ，開発コストや開発期間に余裕がないにもかかわらず，標準化への対応は遅れていました．このような状況下で，ハードウェアや開発環境まで含めた組み込み機器の開発プラットホームの標準化を行い，ソフトウェア部品の流通促進や移植性の向上を目指してT-Engineは生まれました．

T-Engineは，T-Kernelという標準リアルタイムOSで動作します．T-Kernelは，T-Engine上で動作する多くのミドルウェア，アプリケーションの実装プラットホームであり，ユビキタス・コンピューティングの共通カーネルとなります．

T-Engineフォーラムは2002年6月に発足し，国内外の著名なセット・メーカやソフト・ハウスなどが参加し，2003年末には300社を超える大きな組み込みオープン・プラットホーム団体となりました(http://www.t-engine.org/)．

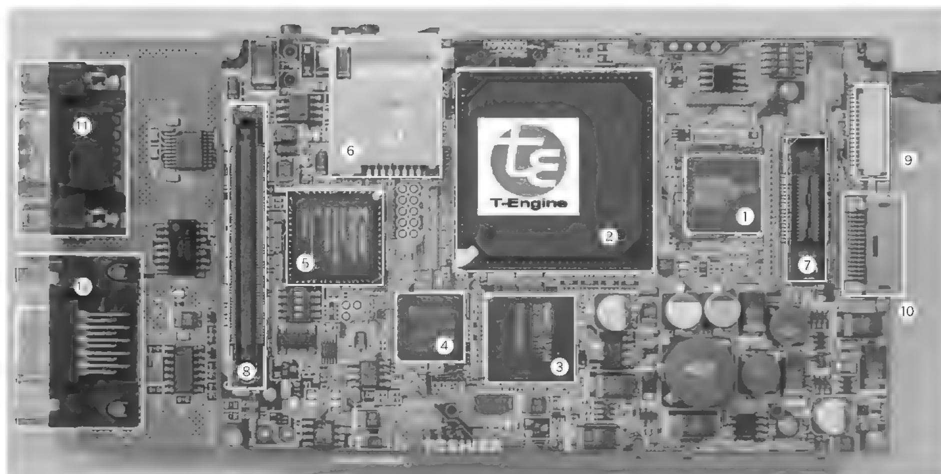
● TX4956の概要

T-Engine/TX4956は，型番からわかるようにTX4956を搭載しています．TX4956はTX49/H4プロセッサ・コアに64ビットのSysADバス・インターフェースをもつ高性能RISCマイクロコントローラです．TX49/H4プロセッサ・コアはMIPSの64ビットRISCアーキテクチャをベースにしたCPUコアです．表5にTX4956の概要を示します．

TX4956は最大動作周波数が400MHzであり，SysADバスは32ビット/64ビットの選択が可能です．プリンタ・ネットワーク機器，セット・トップ・ボックスなど大量のグラフィック・データを扱うアプリケーションや組み込み用途向けに最適なCPUです．

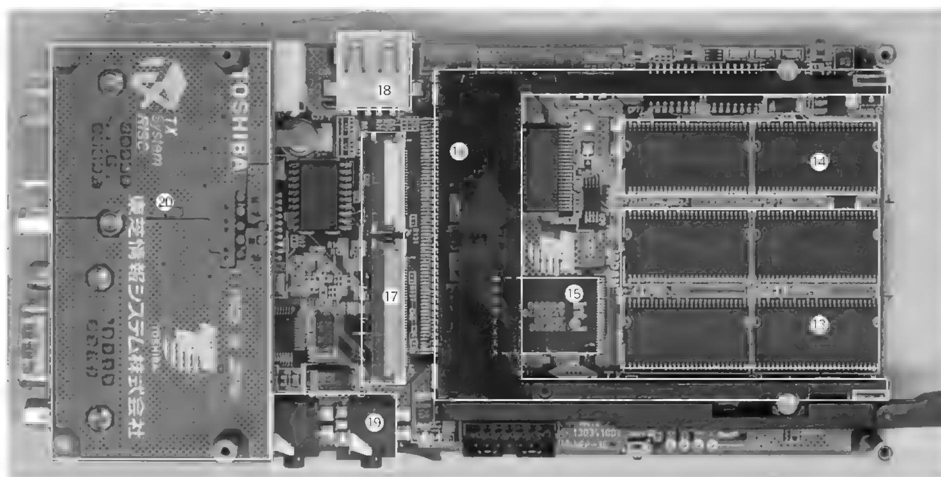
● T-Engine/TX4956の概要

写真2にT-Engine/TX4956の外観を，図7にそのブロック図を示します．T-Engine/TX4956は，TX4956を中心に，標準T-Engine仕様で用意するインターフェースを実現するため，NEC製コンパニオン・チップVRC5477をはじめ，各種コントローラを使用しています．TX4956のSysADバスを32ビット・モードに設定し，VRC5477との接続を行っています．また，外部インターフェースとしてPCIバス，ローカル・バス(16



(a) 表面

①CPU：TX4956-400MHz
②ノース・ブリッジ：VRC5477（NEC）
③表示コントローラ：μPD72255Y（NEC）
④PCMCIAコントローラ：R5C475II（RICOH）
⑤PLD：XC2S150（Xilinx）
⑥eTRONチップ・コネクタ
⑦EJTAGコネクタ
⑧拡張コネクタア
⑨N-WIREコネクタ
⑩シリアル・コネクタ
⑪シリアル・コネクタ
⑫RGBコネクタ
⑬SDRAM
⑭表示VRAM
⑮フラッシュ・メモリ
⑯PCMCIAコネクタ
⑰LCD・タッチ・パネル・コネクタ
⑱USBホスト・コネクタ
⑲AUDIOコネクタ
⑳デバッグ・ボード



(b) 裏面

写真3　T-Engine/TX4956の各部

ビット）を搭載しており，機能拡張にも柔軟に対応可能です．

デバッグ環境としては，EJTAGポートを用意しています．EJTAG対応のICEと接続することで，リアルタイム・デバッグが可能になります．また，オンチップ・デバッグ・サービス・ユニットを使用してリアルタイム・デバッグが可能なN-WIREコネクタも搭載しています．

● T-Engine/TX4956の各部

写真3にT-Engine/TX4956の各部を示します．

T-Engine/TX4956では，TX4956を400MHzで動作させ，チップセットとしてVRC5477を経由してSDRAMを64ビット・バス幅で128Mバイト，フラッシュ・メモリを16ビット・バス幅で16Mバイト接続しています．

さらに標準T-Engineで提供する各種インターフェースを，VRC5477をはじめ，各社LSIで実現しています．

▶eTRONカード・インターフェース（SIMカード・インターフェース）

eTRONカード制御ロジックは，PLDに実装しています．

▶LCDインターフェース

標準T-Engine開発キットであるLCDボードを接続できます．またRGB用出力コネクタを使用することにより，パソコンなどの外部ディスプレイに表示できます．コントローラはNEC製μPD72255Yを搭載しています．

▶リアルタイム・クロック

スーパ・キャパシタによる電源バックアップ機能を搭載しています．

▶PCカード・インターフェース

RICOH製R5C475IIをコントローラとして搭載しています．このコントローラはPCIバスに接続されており，CardBus対応カードも使用可能です．

▶USBホスト・インターフェース

USB 1.1準拠のホスト・コントローラを搭載しています（VRC5477に内蔵）．

▶シリアル・ポート・インターフェース

16550互換のシリアル・コントローラを搭載しています

表6　T-Engine規格の拡張バス専用コネクタのピン配置

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称	ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
1	GND	2	PCICLK[2]	49	GND	50	/FRAME	97	A13	98	A5
3	GND	4	PCICLK[1]	51	GND	52	/DEVSEL	99	A12	100	A4
5	GND	6	PCICLK[0]	53	PAR	54	/SERR	101	A11	102	A3
7	/REQ2	8	VCCIO(3.3V)	55	/CBE[1]	56	PCIAD15	103	A10	104	A2
9	/REQ1	10	VCCIO(3.3V)	57	/CBE[0]	58	PCIAD14	105	A9	106	A1
11	/REQ0	12	/GNT2	59	PCIAD12	60	PCIAD13	107	A8	108	A0
13	GND	14	/GNT1	61	GND	62	PCIAD11	109	WE[1]	110	/WR[0]
15	GND	16	/GNT0	63	GND	64	PCIAD10	111	GND	112	/RD
17	PCIAD31	18	IDSEL[2](PAD29)	65	PCIAD8	66	PCIAD9	113	D15	114	D7
19	PCIAD30	20	IDSEL[1](PAD28)	67	PCIAD6	68	PCIAD7	115	D14	116	D6
21	PCIAD29	22	IDSEL[0](PAD27)	69	PCIAD4	70	PCIAD5	117	D13	118	D5
23	PCIAD27	24	PCIAD28	71	PCIAD2	72	PCIAD3	119	D12	120	D4
25	GND	26	PCIAD26	73	GND	74	PCIAD1	121	D11	122	D3
27	GND	28	PCIAD25	75	GND	76	PCIAD0	123	D10	124	D2
29	PCIAD23	30	PCIAD24	77	/PCIRST	78	LOBAT	125	D9	126	D1
31	PCIAD22	32	Reserve	79	MPOWER	80	/INTA	127	D8	128	D0
33	PCIAD21	34	Reserve	81	WAKEUP	82	/INTB	129	VBAT	130	VBAT
35	PCIAD19	36	PCIAD20	83	/EXT_INT[1]	84	/INTC	131	VBAT	132	VBAT
37	GND	38	PCIAD18	85	/EXT_INT[2]	86	/EXT_INT[0]	133	VBAT	134	VBAT
39	GND	40	PCIAD17	87	A17	88	/CS1	135	VBAT	136	VBAT
41	/CBE[3]	42	PCIAD16	89	A16	90	/CS0	137	GND	138	GND
43	/CBE[2]	44	/STOP	91	A15	92	IRDY	139	GND	140	GND
45	/LOCK	46	/PERR	93	GND	94	A7				
47	/IRDY	48	/TRDY	95	A14	96	A6				

(VRC5477に内蔵).

▶ 音源チップ(音声CODEC)

AC97インターフェースを使用し，ライン入力(ステレオ2チャネル)，ヘッドセット端子(マイク&イヤホン)を搭載しています(VRC5477内蔵のAC97インターフェースを使用).

▶ 拡張バス・インターフェース

T-Engine規格の拡張バス専用コネクタを搭載しています(京セラエルコ社製140ピン・コネクタ).拡張バス・インターフェースには，32ビット/33MHzのPCIおよびローカル・バス(16ビット)を接続可能です(表6).

● T-Engineのミドルウェア～ハードウェアの共通化～

T-Engineフォーラムでは，ミドルウェアを各社CPU用に再コンパイルするだけで，各社製T-Engineボード上で動作させることができるしくみ作りをすすめています．CPUフリー，つまりCPUに特定されることのないミドルウェア提供が，T-Engineの大きな目的です．

東芝情報システムで開発したT-Engineボードは，搭載しているコントローラ・インターフェースをすべてNEC製のT-Engine/V_R5500と同一にしました．これはミドルウェアの流通を促進するための重要な要素と考えています．T-Engine/V_R5500とT-Engine/TX4956は，ハードウェア・レベルで非常に高い互換性をもっています．

さらに，各インターフェースを共通化したことにより，T-Engine/V_R5500およびT-Engine/TX4956の相互間でのミドルウェアのバイナリ互換も実現できるのでは，と考えています(MIPS IV環境でソフトウェア開発を行った場合)．ハードウェアを直接制御するようなミドルウェアに対しても，容易に移植が可能となるので，短期間で高性能なミドルウェアを提供できると考えています．

● ミドルウェアの強化

T-Engine/TX4956上では，さまざまなミドルウェアの評価を容易に行えるようにドライバなどの準備をしています．

たとえばMacromediaが開発した，音声やベクタ・グラフィックス(またはベクトル画像)のアニメーションを組み合わせたWebコンテンツの一種であるFlashを，携帯情報端末でも再生可能にするFlash PlayerやGUI関連，ユビキタス・コンピューティングを実現するための無線LAN通信に対応するため，IEEE802.11a/b/gの各種無線LAN用のドライバやセキュリティ関連のソフトウェアを移植/順次開発しています．

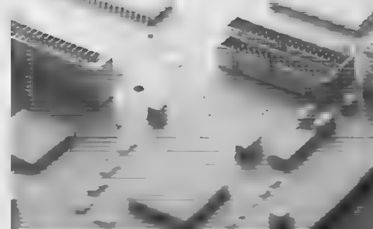
〈黒瀬 浩史〉

よしだ・としや/てらお・たかひろ　(株)東芝
くろせ・こうじ　東芝情報システム(株)

第2部 Appendix 3

FPGA + CPU 構造のプログラマブル・システム・オン・チップを実現する

クイックロジック QuickMIPSの概要



河盛 高史

1 はじめに

現在の組み込み市場は多様に細分化され，その分野ごとに規格，システム要件，市場動向などが異なります．クイックロジックでは新たな市場動向として次の3項目に注目しています．

(1) データ・インスペクション

現在のシステムでは単なるデータ転送に止まらず，データ分析機能を持つことが期待されている．

(2) 高性能化

目的の機能を実現するためにインターフェースと処理速度の高速化が求められる．

(3) デザイン・サイクルの短縮化

システム設計者が半導体メーカにもっとも強く求めることは，新規開発システムの製品化期間を短縮できる製品を提供すること．

特に競争が厳しい状況におかれたシステム設計者は，このような課題に対してさらに限られた開発資源で取り組まなければなりません．

従来このような課題には表Aに挙げた3種類のデバイスによる実装方法がおもに採用されてきました．すなわち，組み込みシステムを実装する特定用途集積回路(ASIC)，特定用途標準製品(ASSP)ならびに極めて性能の高い汎用プロセッサです．しかし，システム設計者がこれらいずれの方法を採用しても，上述の困難な課題を克服できる強力なソリューションが得られない場合もあります．

2 FPGA + CPUというアプローチ

このような課題を解決するためのクイックロジックの戦略は，組み込みシステム開発に取り組む企業の要求を満足できる標準機能，すなわちフィールド・プログラマブル・ゲートアレイ(FPGA)を統合した製品を提供することです．このFPGAが目指しているおもな目的は次の二つです．

一つ目はこのロジックを活用して，カスタム・ロジックとの直結に必要な各種I/Oインターフェースを提供することで特定用途の要求に対応するデバイスをカスタマイズすることです．

二つ目はこのロジックを利用してコントロール・プロセッサの特定タスクを肩代わりするコプロセッサ・エンジンを実装することです．これにより，以下のような利点が得られることになります．

- FPGAロジックは汎用コントロール・プロセッサと比較して，所定のタスクを実行する目的に応じてより高度に最適化できるため，システム性能が向上する
- CPUコア上で開放された帯域幅を利用して追加的な機能を実行できるため，競合製品に対する差別化が増強される
- CPUコアの動作性能と同等の特性で，しかもより低周波のCPUクロックで実行可能なため，電力消費の低減を実現できる．その結果，ファンや放熱板が不要となり，システム・コスト，ならびに信頼性における改善が得られ，さらにネットワーク接続におけるシステムのパワーアップ機能(パワー・オーバEthernetなど)によりシステムの簡略化と柔軟性においても改善が得られる

このアプローチではFPGAファブリックを修正することにより，システムのI/O規格が変更された場合の対応や，新しいデータ処理プロトコルの実装が可能になるため，ASICの特徴(プラットホームの競合によるリバース・エンジニアリングの危険性が低い)を生かしながら，市販のASSPがもたらすシステムの柔軟性を実現できることになります．

このように極めて優れた特徴をもつQuickMIPS製品ですが，費用対効率の面ではどうでしょうか？　FPGA技術を一つのチップ上に集積すれば，チップ・サイズはASICやASSPよりも大きくなるのではないのでしょうか？　確かに純粋な標準セル実装と比較すればFPGAゲートはより大きなチップ領域を占有することになります．しかし，クイックロジックの特許であるViaLink技術は競合製品と比較して大幅に費用対効率に優れた膨大なゲートを提供できま

表A　組み込みシステムの実装方法

技　術	長　所	短　所
ASIC	●デバイスのコストが安価 ●デバイスをアプリケーションの特定要件に最適化 ●競合によるシステム・クローニングはほとんど不可能	●長期のデザイン期間 ●変化し続ける技術規格への柔軟な対応に欠ける ●費用(マスク・コスト)の高騰により，年間使用量が数十万を超えないと実用的でない
ASSP	●迅速な製品化 ●豊富なチップ・ソリューションにより安価	●カスタム・システム・インターフェースへの接続に外部FPGAが必須 ●外部接続ペリフェラルに対するシステム性能の限界 ●付帯作業に対する限定的なCPU帯域幅 ●競合によるデザイン複製が比較的容易
汎用プロセッサ	●システムの柔軟性が高い	●市販FPGAデバイスの追加利用による高額なシステム・コスト ●消費電力の増大 ●競合によるデザイン複製が比較的容易

す．その結果，このSoCの価格は組み込みプロセッサとFPGAを個別に実装した製品と同等でありながら，システムの性能特性をそのまま維持し(すべての機能が同一チップ上に集積されているためチップ間遅延は問題にならない)，さらに消費電力の削減とボード・デザインの簡略化も実現します．

3 QuickMIPS製品概要

すべてのQuickMIPS製品は，特徴的な次の二つの要素で構成されています．

- コンフィグレーション可能な大規模ゲート
- 組み込みアプリケーションには必須となるペリフェラル・コア・セットを特徴とするASIC機能セット

● FPGAセル

ウエハ製造工程の一部となっている極めて信頼性の高い金属間接続技術を利用して，ゲートをプログラム処理するViaLink技術により，FPGAセル部分のチップ・サイズはSRAMベースのFPGAデバイスよりもはるかに小さくなります．

このFPGAは数種類のセルで構成されています．

- 最大36のデュアル・ポートRAMやROM，さらにFIFOを実装可能な2314ビット・メモリ領域．各モジュールは4種類の独立したブロック形状にコンフィグレーション可能で，さらに幅や深さを増大させるためのカスケード処理も可能
- 標準デュアル・レジスタ，ワイド・ファン・インと複数の同時出力機能対応のマルチプレクサ・ベース・ロジック・セル．セルは2個の6入力ANDゲート，4個の2入力ANDゲート，7個の2：1マルチプレクサ，そして非同期SETならびにRESET制御信号付き2個のDタイプ・フリップフロップで構成．またこのセルは6個の出力(4個のコンビナトリアルおよび2個のレジスタ処理)も装備
- 乗算，加算ならびに累算機能内蔵のハード・ワイヤードDSPビルディング・ブロック

このFPGAには4分割可能なクロック・ネットワークと多数のユーザ設定可能なフェーズ・ロックト・ループ(PLL)も搭載されています．

● ASIC機能

QuickMIPSデバイスのASIC機能領域に実装されているおもな機能は以下のとおりです．

▶32ビット MIPS CPUコア

このコアは一般的に普及している各種開発ツールや組み込みOSで幅広くサポートされており，一般的な実証済みソフトウェアの活用や開発作業の軽減が可能となる．メモリ管理機能(MMU)を搭載しているため，Linuxベース・ソフトウェアのソリューションに対応できる．

▶コンフィグレーション可能なメモリ・コントローラ

QuickMIPSデバイスは，モジュールや個別デバイスなど，各種外部SDRAMサブ・システムとの接続が可能で，これによりユーザはその必要条件に最適なコンフィグレーションを選択できる．またこのコントローラでは8，16，32ビット・クラスのSRAMやROM，フラッシュ・メモリなど多数の低速メモリ・ペリフェラルとの直結が可能となっている．

▶32ビット PCI

PCI規格バージョン2.2に準拠するこのインターフェースは，最高クロック周波数66MHzで動作するため，低価格なネットワーキング，ストレージ，グラフィックス・ペリフェラルなど，広範囲な選択肢が得られる．また，最新I/O技術では大規模なパソコン市場に対応するPCIが採用されているため，この機能によってシステムの柔軟性がさらに増強される．

▶Ethernetインターフェース

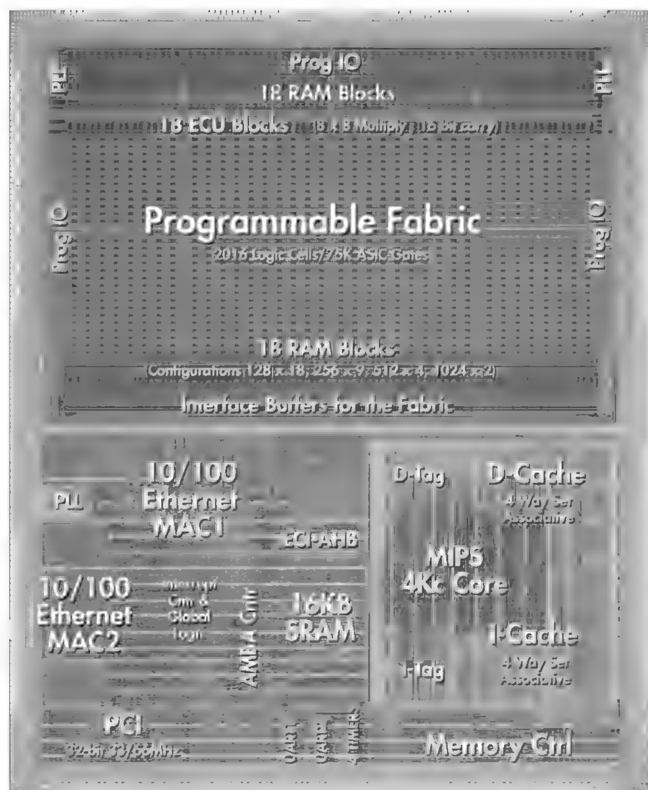
遠隔管理を実現するためにネットワークへのシステム接続が必須条件となったため，このインターフェースが実現されたが，現在これは大半の組み込みアプリケーションにとって事実上不可欠となっている．PCIベースEthernetコントローラではPCI帯域幅がほかのペリフェラル用にリザーブされるため，このインターフェースをチップ上に搭載したことによりシステム・スループットが向上する．EthernetコントローラではでPCIインターフェース固有の遅延問題から解放され，ダイレクト・メモリ・アクセス(DMA)などのオンチップ・リソースも活用できることになる．

このデバイスでは，メディア非依存型インターフェース(MII)ポートをオフチップで提供しているので，コントローラに対する外

表B　QuickMIPSデバイス一覧

特　性	デバイス			
	QL901M	QL902M	QL903M	QL904M
CPU動作周波数(MHz)	133	175，200，233	175，200，233	175，200，233
最大メモリ・バス周波数(MHz)	66.5	116.5	116.5	116.5
サポート対象の SDRAMメモリ・タイプ	×8，×16，×32 コンフィグレーション	×8，×16，×32 コンフィグレーション	×8，×16，×32 コンフィグレーション	×8，×16，×32 コンフィグレーション
CPUパイプライン/ ローカル・バス・クロック比	1：2	1：2，1：3，1：4	1：2，1：3，1：4	1：2，1：3，1：4
FPGAファブリック規模 (マクロセル数/システム・ゲート数)	2016/575	2016/575	1152/300	1152/300
I/Oピン数	252(合計)	122	94	109
PCIコントローラ	あり	あり	あり	なし
Ethernetインターフェース	2	2	2	1
パッケージ・サイズ	680 BGA	544 BGA	544 BGA	544 BGA

Ethernetポート未使用時(QL902Mデバイス採用でシステム上にEthernetポートが不要な場合，最大152個のI/Oポートの利用が可能)MIIポート1個当り追加15ピン利用可能．



写真A　QL902Mチップ

部スイッチ・ファブリック，Ethernet物理層(PHY)，さらにそのほかI/Oペリフェラルとの接続が可能になる．

▶ 汎用ペリフェラル

すべてのQuickMIPSデバイスにはシリアル・ポート，タイマ，ならびに割り込みコントローラが搭載されている．

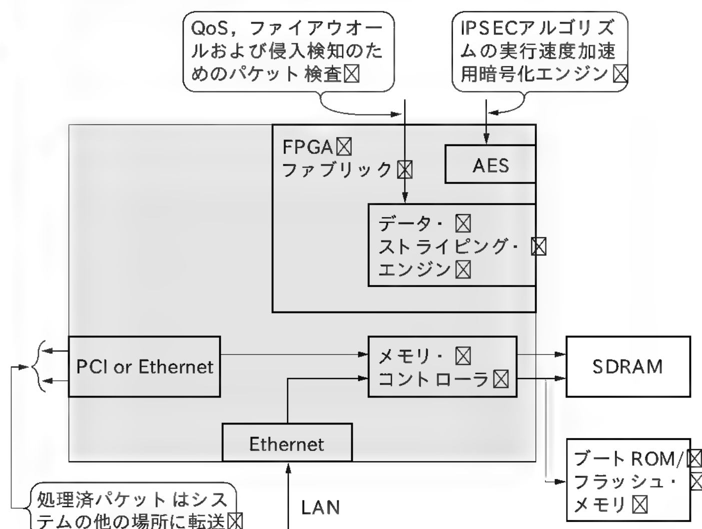
各種組み込みアプリケーションはそれぞれ異なります．コプロセッサ・タイプの機能を実装するための大規模FPGAファブリックを必要とするアプリケーションもあれば，カスタムI/Oへの直結機能を供給する目的で小規模リソースのみを必要とするものもあります．また，顧客によってはシステム機能セットが決定され，変更の必要がない場合，そのシステムにPCIを搭載する必要性もなくなります．

これらの各種要件に対応するため，表Bに示すような各種のデバイスを用意しています．これら製品はすべてソフトウェアの互換性を保っているため，異なるQuickMIPSデバイスが採用されている状況でも，ユーザはさまざまなシステム間で共通のアプリケーション・ソフトウェアを活用できることになります．写真AにQL902Mチップの写真を示します．

4 QuickMIPSのアプリケーション事例

● VPN駆動型ゲートウェイの例

図AにQuickMIPSを使ったVPN駆動型ゲートウェイのブロック図を示します．この例では，プログラマブル・ロジックはパケット・ヘッダとペイロード・データの分析を行うように設計されています．これによりシステムは受信するビット・ストリームの特定



図A　QuickMIPSベースのVPN駆動型ゲートウェイのブロック図

データ・ビットから読み込まれた情報に基づいて配信方法を決定できるため，通常のデータ・ストリームに先んじて音声および映像送信の優先順位付けに利用できることになります．このような音声/映像メディアでは，ネットワークに起因する遅延についてのユーザの許容度は極めて狭いため，この機能は重要なものになります．

このアプローチでは，ASICソリューション固有の機能(すなわち競合によるプラットホームのリバース・エンジニアリングの危険性を低減)を発揮する一方で，市販のASSPの特徴であるシステムの柔軟性も得られるため，そのプログラマブル・ロジックを修正することにより，I/O規格の頻繁な変更に対する対応や，新しいデータ処理プロトコルの実装が可能になります．

● 知的資産の保護

ViaLink技術をベースにしたシングルチップ・ソリューションを採用することで得られるもう一つの重要な利点は，知的資産(IP)をリバース・エンジニアリングから守ることです．具体的には次のような点があげられます．

(1) プログラマブル領域にはコンフィグレーションROMが不要なため，解析の対象となるようなビット・ストリームが存在しない．電子顕微鏡を用いたとしてもアレイ上で接合部と非接合部とを区別することは不可能．
(2) プログラマブル領域と組み込みプロセッサ・システムがシングル・チップ上に集積されているためオフセット・アクセスの頻度も低いものとなる

この市場分野ではセキュリティについての関心がますます高くなっています．帯域幅に対する要求が高まり，さらにAESなど具体的なセキュリティ規格が成熟するにともない，暗号化やハッシュ・コーディングなどの作業をCPUコアから解放するために，これをハードウェア・ブロックに実装させるという傾向が強まっています．

かわもり・たかし　クイックロジック(株)

第2部 Appendix 4

ディジタル・ホーム・ネットワーク・アプリケーション向け

IDT RC32434 Interprise 総合コミュニケーション・プロセッサ



Matt Jones

RC32434 Interpriseとは

米国Integrated Device Technology(IDT)社のRC32434 Interprise統合コミュニケーション・プロセッサは，メディア・アダプタやメディア・サーバ，IPネットワークを利用したマルチメディア機器などのディジタル・ホーム・ネットワーク・アプリケーション向けのプロセッサです．

CPUコアには，最大で400MHzで動作する32ビット MIPS32 4Kcコアを採用しています．これは現在，出荷されているMIPSコアのなかでは最高速度のものです．

また，本プロセッサは，次の機能ブロックを内蔵しています．

- ×16DDRメモリ・コントローラ
- 32ビット PCI 2.2インターフェース
- 10/100MbpsのEthernetポート
- 専用のローカル・メモリI/Oコントローラ
- 64バイトのNVRAMとAU(認証ユニット)

以下では，RC32434の概要について解説します．

● 400MHzのMIPS 4Kcコア

図AにRC32434のブロック図を示します．

本プロセッサは，MIPS32 4Kcコアを採用しており，このコアは最大400MHzで動作します．

RC32434では，動作周波数がそれぞれ266，300，350，400MHzの四つのバージョンが用意されています．

このコアの性能により，本プロセッサはEthernetやIEEE802.11a/b/gのようなネットワーキング・プロトコルをサポートすることができます．さらに，将来，セキュリティのアップグレードを行いたい場合や，ユニバーサル・プラグ＆プレイ(UPnP)のようなプロトコルを使用することになった場合にも，これらを組み込めるだけの十分な余裕を持っています．

● ×16DDRメモリ・コントローラ

本プロセッサでは，同じメモリ帯域幅をもつSDRAMベースのサブシステムに比べて2倍のメモリ帯域幅をもつDouble Data Rate(DDR)メモリ・コントローラを採用しています．PCメーカが出荷しているDDRメモリを採用することで，コストの削減にもつながります．本プロセッサのDDRメモリ・コントローラは，×16の外部メモリ構成に対応しており，最大256MバイトのDDR-SDRAMをサポートしています．

また，このDDRメモリ・コントローラ・サブシステムでは，ローカル・メモリに接続されるバスとI/Oデバイスに接続されるバスが分離されています．その理由は，アドレスとデータが多重化されたバスから得られる利点以上に，DDRの帯域幅の利点のほうが大きいためです．

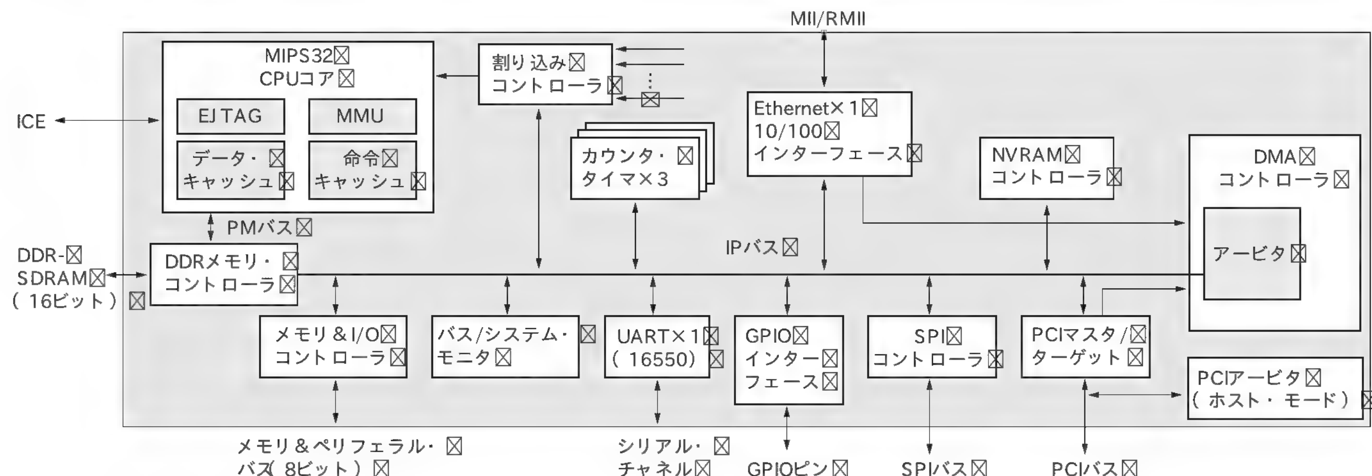
● PCIインターフェース

本プロセッサは，32ビット PCI Rev.2.2のインターフェースを内蔵し，6個のバス・マスタに対応しています．

本プロセッサのPCIバス・インターフェースは最大66MHzで動作し，バースト転送用に最適化されています．また，166 Mbpsのスループットを維持するように設計されています．4組の256バイト・レジスタとデータ・バッファリングの採用により，プロセッサが頻繁にデータをアクセスするときでも，このデータ・バーストを維持することができます．

● Ethernetインターフェース

本プロセッサは，10/100MbpsのEthernetインターフェースを備えています．標準のMII(Media Independent Interface)またはRMII(Reduced MII)のオプションにより，広範囲なネットワーキング・デバイスに接続できます．Ethernetポートでは，専用のDMAチャネルなどのオンチップ・リソースを利用することができます．また，



図A　RC32434のブロック図

各MACアドレスをサポートする2系統の512バイトの送信FIFOと受信FIFOや，着信アドレスをチェックするアドレス検出ロジックを備えています．

● ダイレクト・メモリ・アクセス(DMA)

本プロセッサは，6チャネルの専用DMAを内蔵しています．これらのうちの2チャネルはPCIコントローラ用です．別の2チャネルはEthernetインターフェース用，さらにほかの2チャネルはメモリ間転送用です．6チャネルすべてがフライ・バイ転送となっているため，コア・プロセッサによるストア動作やパス・スルー動作が不要です．

● NVRAMと認証ユニット

64バイトの内蔵不揮発性ランダム・アクセス・メモリ(NVRAM)とセキュリティ機能用の認証ユニット(AU)を備えています．これにより，コンテンツ保護やディジタル権利管理のためのアプリケーションなどを組み込むことができます．

● クロック

本プロセッサでは，次の二つの方法により，ハードウェア設計に対するシステム・クロック・ツリー設計の簡素化が図れます．

1) さまざまなインターフェースを駆動するために必要なクロック信号を発生させるために，一連のPLLを内蔵している．さらに高周波回路がデバイスに内蔵されているため，EMI放射の削減が図れる．実際，コアの最大動作周波数は400MHzだが，プロセッサを駆動するために必要な動作周波数は25MHzだけ．
2) 本プロセッサは，DDRバスやローカル・メモリ・バスなどの外部バスに対してクロック信号を供給することができる．同期PCIバス動作の場合，ローカル・バスをPCIクロック・ツリーに接続して，このインターフェースを駆動することができる．これにより，外付けのチップを減らして，ボード・レイアウトを簡素化することができる．

● 消費電力，パッケージ

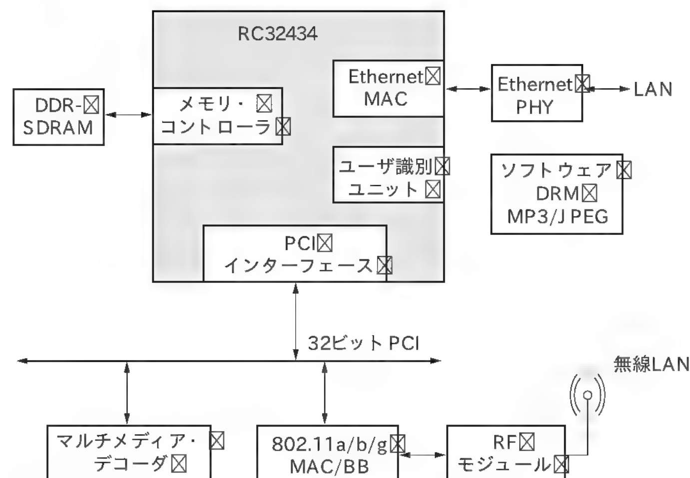
本プロセッサは0.13μmのCMOSプロセスで製造され，接続に6層メタルを使っています．動作電圧はそれぞれ，コア部分が1.1V，DDRメモリ・サブシステム部分が2.5V，I/Oリング部分が3.3Vとなっています．ワースト・ケースの消費電力は2.5Wに制限されています．パッケージは256ボールのBGAパッケージを採用しています．

RC32434の応用例

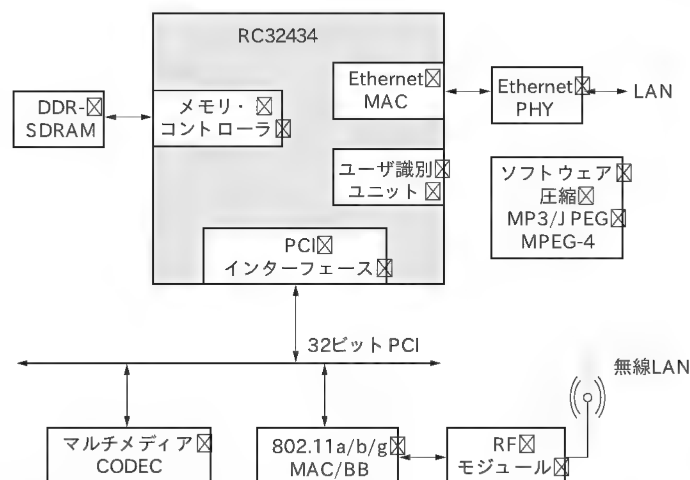
RC32434はディジタル・ホーム・ネットワーク・アプリケーション向けのプロセッサです．本プロセッサの応用例として，メディア・アダプタとIPビデオ監視カメラについて紹介します．

● メディア・アダプタ

メディア・アダプタ(図B)は，ディジタル・ホーム・ネットワークでストリーミングされるマルチメディア・コンテンツの表示に使用される端末です．図Bでは，メディア・アダプタが本プロセッサ内蔵のEthernet MACを利用して，ディジタル・ホーム・ネットワークに接続され，PCIインターフェースを使って，WLANやマルチメディア・デコーダ・チップ・セットのようなペリフェラルが接続されています．CPUはTCP/IP，RTSP，RTP/RTCP，UPnPなどのネットワーキング・プロトコルを実行して，LANとWANを経由してクライアントとして交信します．さらに，このデバイスはPCIバス上の外付けマルチメディア・デコーダ・チップ・セットにあるグラフィックス・エンジンを使ってGUIを動作させます．さらに，内蔵のAU



図B　メディア・アダプタの構成図



図C　IPビデオ監視カメラの構成図

(ユーザ識別ユニット)とチップ内に保存されているキーを使って，DRM方式に対応させています．

● IPビデオ監視カメラ

IPビデオ監視カメラ(図C)では，ビデオのキャプチャ機能，圧縮されたディジタル・ストリームの生成機能，LANまたはWANを経由した転送機能が要求されます．さらに，パンニング，ティルティング，ズーミングなどのリモート・コントロール機能も使われます．ネットワーク接続に対しては，本プロセッサ内蔵のEthernet MACまたはPCIベースの802.11xチップ・セットを使うことができます．また，圧縮されたディジタル・ビデオ・ストリームを生成するビデオ・エンコーダを接続するためにPCIバスを使うこともできます．本プロセッサは，LANに接続するUPnP，ビデオ・ストリームを暗号化するIPSec，さらにはモーション検出モジュールからのトリガによりキャプチャされた画像を送信するメール・サーバなどの，多くのネットワーキング・プロトコルやセキュリティ・プロトコルを処理します．さらに，このカメラはWebベースのマネージメント設定をサポートする必要があります．

マット・ジョーンズ　IDT, Inc.

組み込みプログラミング・ノウハウ入門(第16回)

アクティブ・オブジェクト・モデリングのこころ

ふるまいの合成と検証

藤倉 俊幸

携帯電話のバグがまた報道された注1．電源を入れて待受画面にアンテナ・マークが表示される前に端末を折りたたんだり，圏外に出たり入ったりした場合に，メールの自動受信ができなくなるというものだ．並列イベントの順番によって特定の機能が使えなくなる現象である．バグの原因についてはよくわからないが，部分的なデッドロックが原因ではないかと推測される．

複雑な並列イベントを扱おうとすると，さまざまな問題が発生する．この連載の第13回目(2004年4月号掲載)では，踏み切り制御の問題をイベント対(つい)に分解して，その後に再合成してプログラムを作成する例を示した．その際に，電車が複数走っている場合の扱いは省略してしまった．電車が複数の場合(より一般化していえばアクタに多重度がある場合)には，ハッセ図で表せるようなAND条件だけでなく，ステート・マシンで表すことが必要なOR条件も扱わなければならない．

今回は，単純な順序集合とならない場合にどのようにアクティブ・オブジェクトを作っていくのかを説明する．シナリオを追加しながらアクティブ・オブジェクトを作っていく方法は，前回の$f(a+b) \leqq f(a)+f(b)$で説明した方法を発展させたものでもある．

1 再考・踏み切りモデル

まず，踏み切りモデルをレビューする．再掲になるが，システムの構成は図1のようになっていた．電車がセンサAを通過すると踏み切りの遮断機が下りて，センサBを通過すると遮断機が上がる．

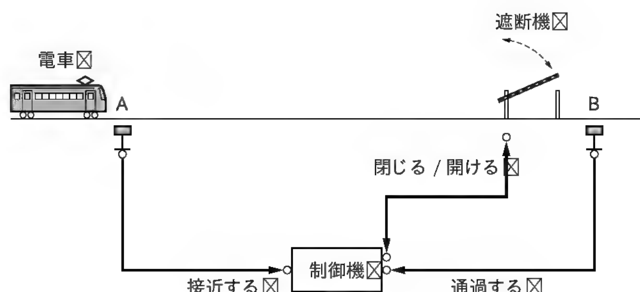


図1　踏み切りモデル

この構成からイベント対を抽出して，イベント間の順序を表すハッセ図を作成した．そしてそのハッセ図から制御機のステート・マシンを生成した(図2)．しかし，図2のステート・マシンでは，複数台の電車が走る場合に不具合が発生する．たとえば，

1) 連続して2台の電車がセンサAを通過した場合，このステート・マシンは2台目の接近イベントを受信できない
2) 1台目がセンサBを通過した際に，遮断機が開いてしまうことがある

の二つがある．1)は大した問題ではない．本当に問題なのは2)である．このときに2台目の電車はまだ踏み切りを通過していない場合もあるので，事故につながる可能性がある．

● 一般的な対応方法

一般に，開発が終了してからこのような不具合がわかった場合には，テスト済みのプログラムはそのままにして，中間にフィルタを設けるだけで対応できる．たとえば，Rose RealTime

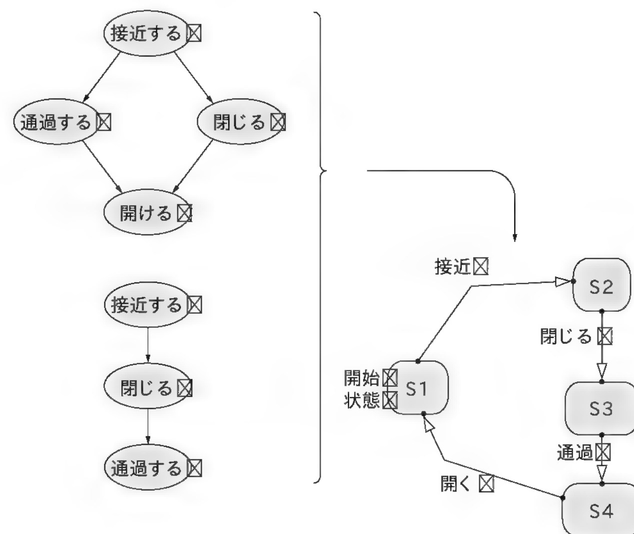


図2　生成されたハッセ図とステート・マシン

注1：http://www.nttdocomo.co.jp/info/customer/caution_s/f900i.html

の場合(図3)は，アクティブ・オブジェクトとインターフェースであるポートが明確に分かれているので，このような対応は比較的簡単である．図3は，Rose RealTimeのアクティブ・オブジェクトのコラボレーションを表現している．物理オブジェクトであるセンサや遮断機に対応したアクティブ・オブジェクトと，踏み切りの制御を行う制御機がポートとコネクタにより接続され，接近や通過などのメッセージが送受信される．そのメッセージの送受信をトリガとして，制御機カプセルの中にある図2のステート・マシンが起動される．

この場合，複数の電車に対応するためにはセンサAとセンサBのポートにトラップを入れて電車の数を制御することになる．具体的には，図3の構造はまったく変えずに，制御機カプセルの内部構造を変更する．この方が変更の影響が局所的で済む．

この変更で実際に対応した例を図4に示す．`TrainCounter`と呼ぶカプセルを新たに制御機カプセルの中に設けて，センサAから電車接近のメッセージが到着すると内部のカウンタを+1する．このカウンタは踏み切り区間内の電車の台数を表すことになる．最初の電車が接近した場合にのみ，`TrainCounter`カプセルは制御機カプセルに電車接近メッセージをフォワードする．一方，センサBから電車通過メッセージが到着した場合は内部のカウンタを－1する．そして，内部カウンタ値がゼロになったときのみ電車通過メッセージを制御機カプセルにフォワードする．こうすることで，既存の開発成果物に手を加える

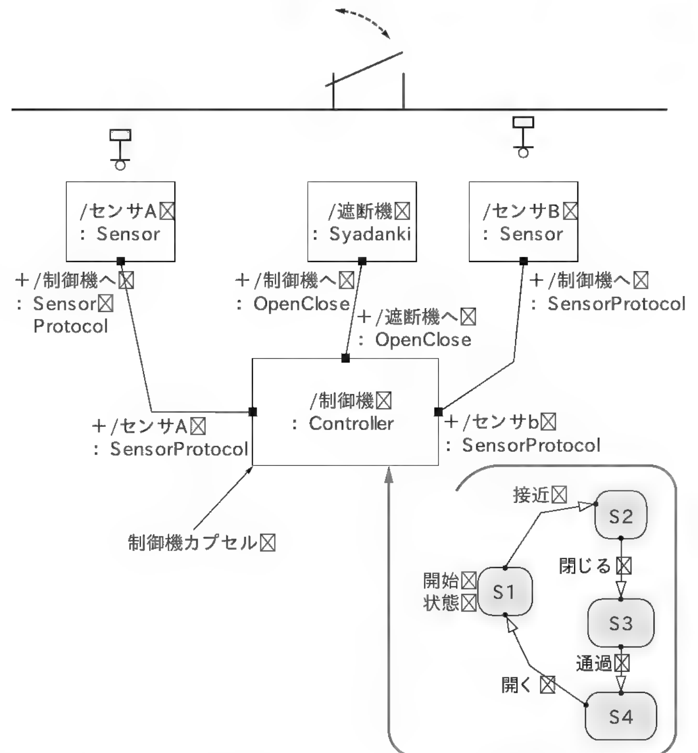


図3　Rose RealTimeの場合

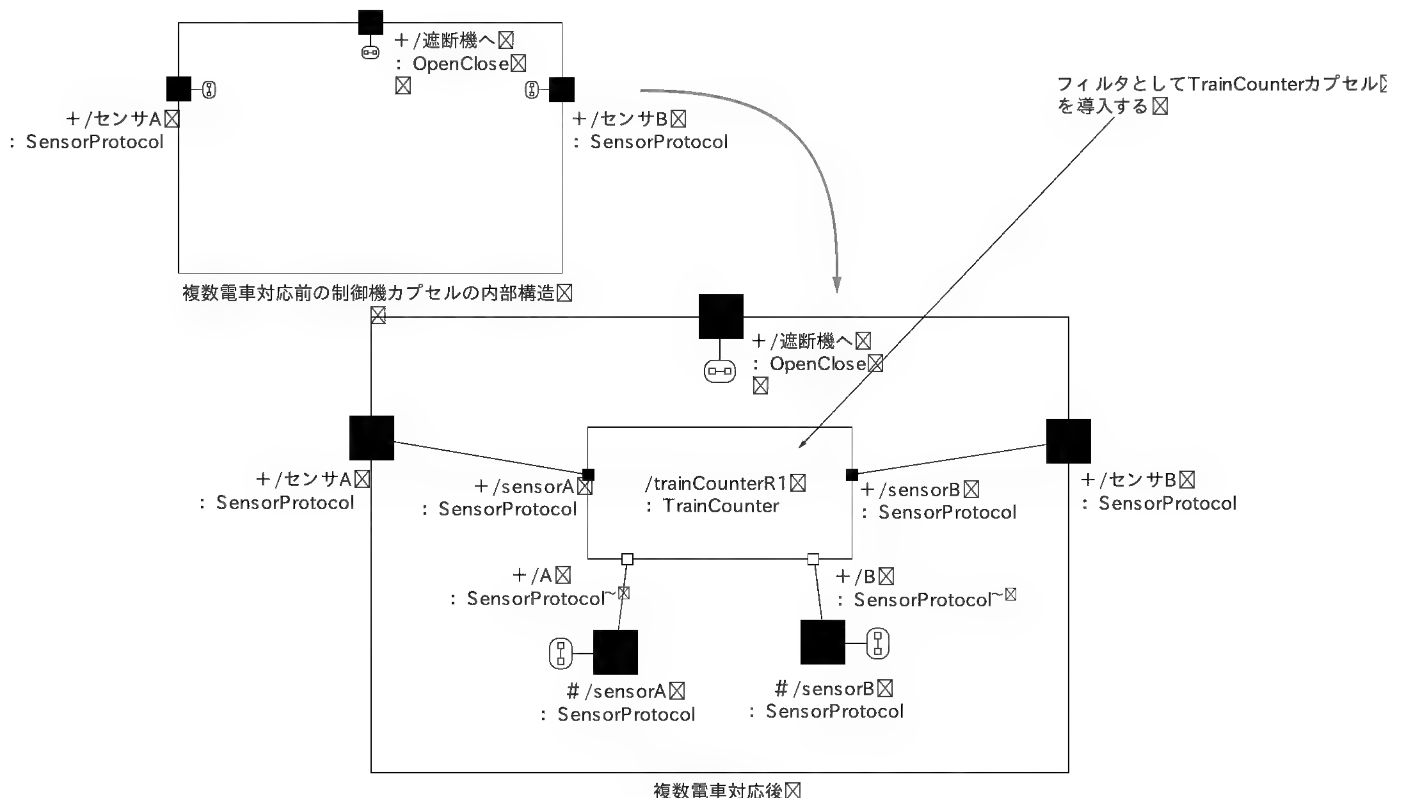


図4　複数電車への対応

ことなく複数電車に対応できる．

これは，「インターフェースとふるまいを分けている」こと，「単機能のクラスでソフトウェアを構成している」ことなど，オブジェクト指向のメリットの一つである．

● ふるまいを合成する場合

では，オブジェクト指向を使っていない場合や，リソースが少なくてクラスの数を増やしたくない場合，またはタイム・クリティカルなアプリケーションなのでメッセージのフォワードを行いたくないような場合には，どのような対応があるだろうか？　このような場合，普通は制御機カプセルのふるまいを直接変更することになる．つまり，図2のステート・マシンを変更するのである．次にこの方法について説明する．

電車が2台の場合のシーケンス図の例を図5に示す．電車は実世界の物理オブジェクトなので，それぞれ並列に動作する．このため，タイミングの異なるシナリオが複数存在することになる（図6）．そして，それらのシナリオの中に2台目の電車がまだ踏み切りを通過していないのに遮断機を上げてしまうものが含まれている．

この場合の電車についての事象対（じしょうつい）は「接近する」→「通過する」が電車ごとに抽出される．これらの事象をプロセス代数式で表して並列化すると，図7の一番上のステート・マシンを得ることができる．ここでは，個々の電車を識別する必要はないので，in1事象とin2事象は同一視することができる．Out事象についても同様なので，図7の真ん中のステート・マシンが導かれる．この真ん中のステート・マシンをよく見ると，状態1と状態3

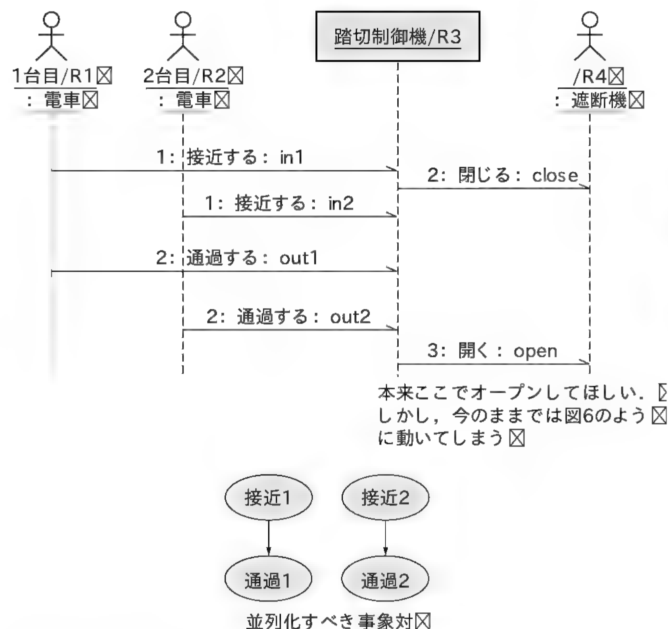


図5　電車が2台の場合

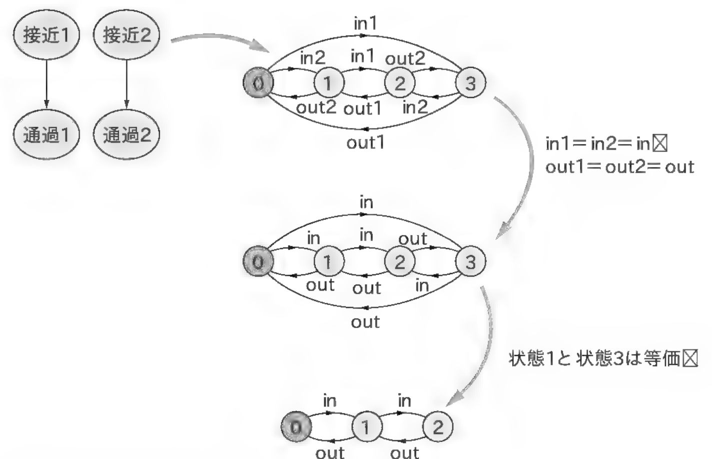


図7　2台の電車の動きを表すステート・マシン

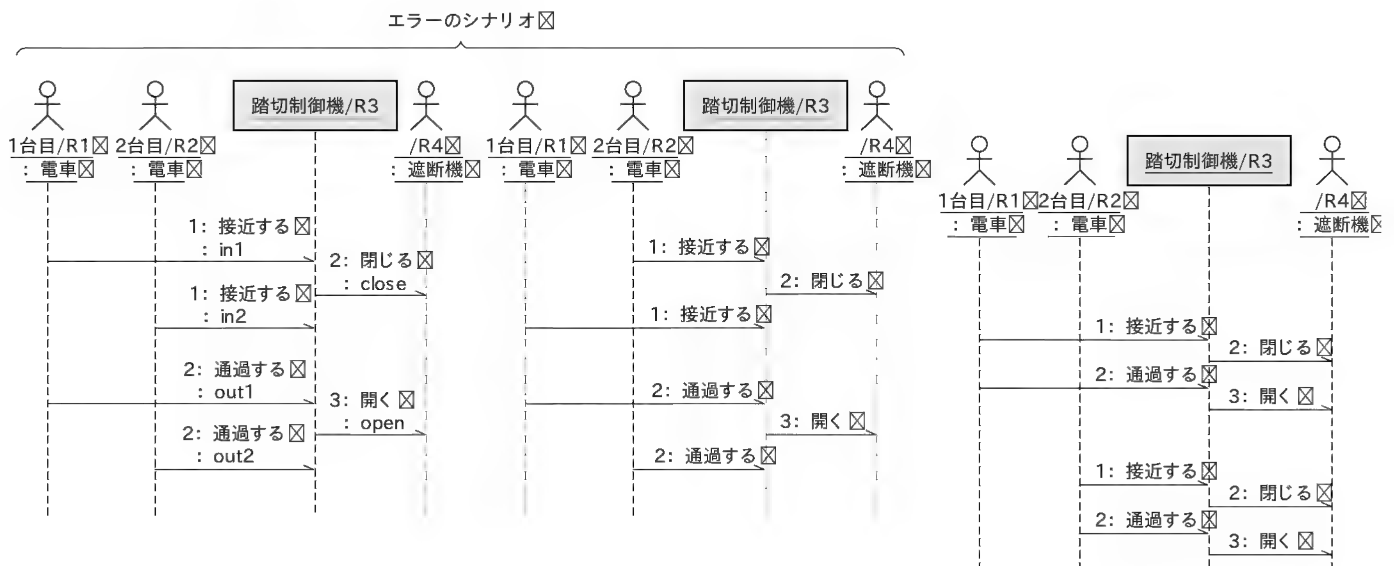


図6　タイミングが異なる別のシナリオ

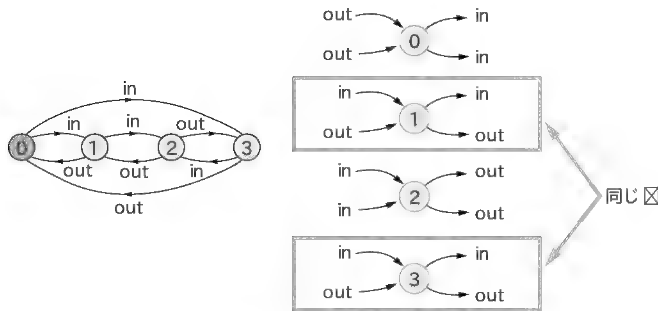


図8　ステート・マシンのリダクション

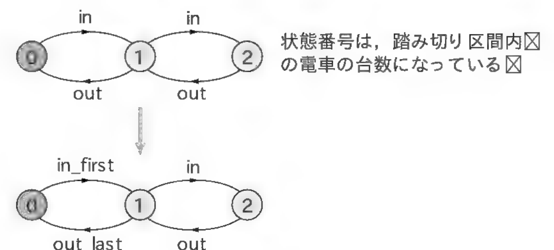


図9　電車ステート・マシンの変更

は遷移の種類がまったく同一なので（図8），識別する必要がないことがわかる．そこでこの二つの状態を重ね合わせてしまうと，図7の一番下のステート・マシンを得ることができる．

システム全体のふるまいを合成する場合は，ハッセ図ではなく，このステート・マシンを使用する．このステート・マシンで電車のふるまいを記述できることになる．冒頭でOR条件を記述するためにステート・マシンを使用すると書いたが，その意味は，「1台目の電車が踏み切り区間に入った後には，1台目の電車が出て行くか，または2台目の電車が入ってくるかのどちらかの事象が起こる」という意味である．どちらか一方しか起こらないので，ORではなく，厳密には排他的ORである．このような途中で分岐が発生するふるまいはハッセ図では表現できないため，ステート・マシンを使用する．

2 制御仕様の検討

電車の動きを記述できたので，次に制御仕様を記述する．今までの仕様は「電車が接近したら遮断機を閉じて，通過したら遮断機を開ける」というものであった．しかし，この仕様に従って動作させるとエラーが発生する．

まず，前半の「電車が接近したら遮断機を閉じる」だが，遮断機は「閉じる」→「開ける」を繰り返すものであるから，閉じたら一度開けてからでなければ閉じられない．1)の問題に対応するために，すでに閉じている状態では，「閉じる」メッセージを受け取っても無視するようにすることも考えられるが，並列動作を考えるとプログラムが複雑化してしまう．また，このような動作は実装の詳細レベルでローカルに考えるべきことであり，システム全体の仕様を考えているときに考慮すると，本質的なシステムの動きを見落としてしまう可能性があるので，ここでは考えない．

そこでここでは，「最初の電車が接近したら遮断機を閉じる」と仕様の解釈を変更する．この仕様の意図は「最初の電車が接近したときのみ遮断機にcloseメッセージを送信する」ということである．受信した側で余計なメッセージを捨てるのではなく，送信側が余計なメッセージを送信しないようにするということである．

次に後半の「電車が通過したら遮断機を開ける」だが，これも前半同様に複数の電車が並列に動作している環境では余計なopenメッセージを送信する可能性がある．さらに問題なのは，最初の電車が通過しただけで遮断機を開けてしまうことである．そこで，ここでは「最後の電車が通過したら遮断機を開ける」と解釈する．

仕様の解釈変更にともなって，電車の動作の中から遮断機を動作させるトリガを取り出さなければならない．電車のステート・マシンを良く見ると，状態番号が踏み切り区間に存在している電車の台数に対応していることがわかる．したがって，最初の電車の接近は，状態0から状態1へのinイベントであることがわかる．それでこのイベントをin_firstと名前を変更して，2台目のinイベントと区別できるようにしよう．outイベントについても同様である（図9）．対応するプロセス代数式は以下のようになる．

```
// 電車
A0=(in_first->A1), A1=(in->out->A1|out_last->A0)
                                    ……………………(1)
```

この連載で使用しているフリーのLTSAツール注2を使用してステート・マシンを直接プロセス代数式によって記述するには，分岐演算子|とローカル状態（この場合A1）を使わなければならない．上記の式は，初期状態のA0からin_firstイベントによりA1状態に遷移する．A1状態では，inイベントまたはout_lastイベントを受理できる．out_lastイベントを受け取った場合には初期状態（A0）に戻る．

準備ができたので，制御仕様をプロセス代数式で記述すると，

```
// 制御仕様-1
B1 =(in_first->close->out_last->open->B1)
                                    ……………………(2)
```

となる．この式は，図2のハッセ図で「接近する」をin_first，「通過する」をout_last，「閉じる」をclose，「開く」をopenで置き換えただけである．上記の式（1），式（2）を並列化オペレータ||で結合するとシステムのふるまいを得ることができる．

```
||F0 =(A0||B1)          ……………………………(3)
```

注2：http://www.doc.ic.ac.uk/~jnm/book/ltsa/LTSA.html

実際にF0のふるまいを合成してみると，図10が得られる．

F0ステート・マシンを良く見ると，遮断機を閉じる前に2台目の電車が通過するパスが存在していることに気付く．遮断機が閉じる前に電車が通過してしまう不具合は，第13回のときに，

Sp4 =(in->close->out->open->Sp4)　　·············(4)

という形で解決したが，この式を1台目の電車専用の式(2)にしてしまったので2台目の電車がすり抜けてしまったのである．

図10のF0ステート・マシンは，実際に実装されるステート・マシンとは意味が異なる．実際のステート・マシンでは，in_firstイベントをトリガとする遷移のアクションとしてcloseイベントが実行される．したがって，F0ステート・マシンの状態1と状態2は実装されたステート・マシンでは一つの状態になる．別のいい方をすると，in_firstイベントとcloseイベントは実装ステート・マシンの動作セマンティックス上はアトミックに処理される．同様に状態0と状態3も一つの状態になる．状態5から状態4への遷移でのcloseイベントは状態1と状態2がいっしょになったことで何もする必要がなくなるので，状態5と状態4も一つの状態にすることができる．したがって，メッセージのキューイングを行ってくれるフレームワークが利用できる場合には図11のようなステート・マシンで実装できる．

closeイベントはプログラムが作り出すイベントなので，in_firstイベントとcloseイベントを不可分に処理するように実装することは可能であるが，in_firstイベントとinイベントは外部イベントなので制御の外になる．そのため，ほとんど同時に発生する可能性があることに気を付けなければならない．たとえば線路が複々線になっている場合である．この場合，close処理中に次の電車によってclose処理が多重に起動されるか，inイベントが取りこぼされるかもしれない．したがって，ステート・マシンがメッセージをキューイングしない場合には，ステート・マシンをトリガする関数がリエントラントかどうかが重要になってくる．

関数によるトリガでステート・マシンを実装する場合，in_firstとinイベントで同一の関数呼び出しを利用できなければステート・マシンを導入する意味がない．図11のステート・マシンでin_first遷移を実行中にin遷移を実行することは，一般にはステート・マシンの動作セマンティックスに反するのでやめたほうが良い．in_first遷移中はまだ，Close状態にはなっていないのでin遷移は起動されず，多重呼び出しとなったイベント関数の挙動は不定になる．ステート・マシンの解釈としては遷移処理中はその外側の状態に属していると考えることが一般的であるため，図12のようなステート・マシンを利用すればin遷移をin_first遷移中に受け付けられることになる．このようにすればイベントの取りこぼしは避けられる．

しかし，inイベント処理中にoutイベントも同時に入ってくる可能性も考慮しなければならないので，キューイングなしの実装メカニズムで並列事象を処理するステート・マシンを実装することはかなりの危険をともなうかもしれない．このことは電車が2台だけでは顕在化しないが，2台がほぼ同時に踏み切り区間に入ってきて，またほぼ同時にすでに踏み切り区間に入っていた別の電車が出た場合に起こりうる(図13)．複数のスレッドから呼び出される静的クラスによるステート・マシン，別のいい方をすると，関数呼び出しでトリガされるステート・マシンをスレッド境界に置く場合には，システム挙動を十分に検証する必要がある．

● 形式仕様導入の目的

図10のようなステート・マシンを作る目的は，プログラムを設計するためではなく，どのような仕様にするべきかを検討

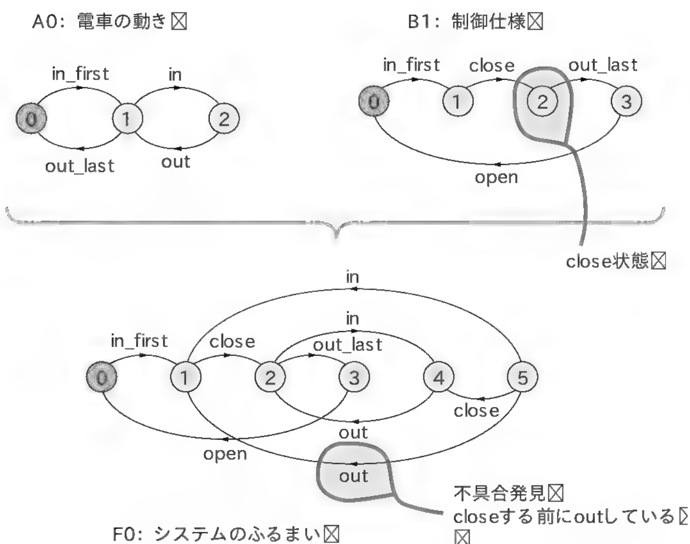


図10　システムふるまいの合成

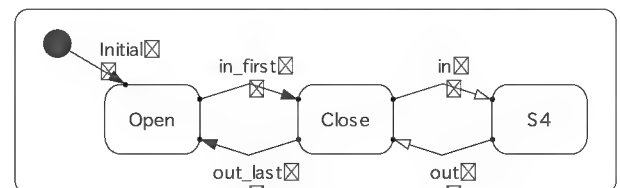


図11　F0ステート・マシンの実装(その1)

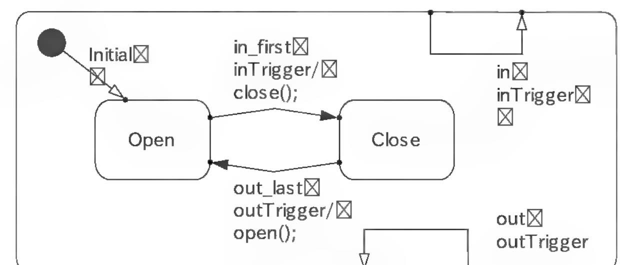


図12　F0ステート・マシンの実装(その2)

するためである．どのような仕様にするとシステムの挙動がどう変わるかを明らかにすることで，どのようなテスト・ケースが必要か，設計時にどのような点に注意すべきかを明らかにすることが目的である．したがって，「遮断機をcloseする前に，2台目の電車が通過してしまうふるまいがある」という現状のF0ステート・マシンに対する正しい対応は，どのように実装でこのリスクを回避するかを検討することではなく，その前にまずこの動きを分析することなのである．

● 正確なふるまいモデルを作る

そこで，F0ステート・マシンの問題の実行パスを仕様的に回避することを考えよう．単純に考えると，closeしてから通過するという条件をシステムふるまいに追加することである．もちろん，この条件は，電車の速度やセンサと踏切の位置などによって物理的に保障できるという前提に基づかなければならない．具体的には，次のプロセス代数式を追加する．

B1c =（close->out->B1c）　……………………（5）

実は，これはうまくいかない．この式を加えてステート・マシンを生成すると図14のようになる．このステート・マシンF0bはデッドロックを起こしてしまうのである．

たしかに，closeする前にoutする実行パスは取り除かれた．しかし，以下の実行系列の場合にデッドロックしてしまう．

in_first
in
close
out
in

状況としては，「最初に踏み切り区間に入った電車が踏み切り区間から出る前に，別の電車が通過して，さらに別の電車が踏み切り区間に入った場合，outイベントを受け取れなくなってしまう」というものだ．結果としては，開かずの踏み切りになってしまう．式（5）を導入したことにより，outするためにはcloseしなければならない．しかし，すでにcloseされていて，制御仕様どおりにふるまうとすれば，いったん遮断機をopenしてからでないとcloseできない．よってデッドロックが発生してしまう．制御システムはデッドロックになるが，実在する物理オブジェクトの電車やセンサは動き続けるので，システムはoutイベントやinイベントを受理できないというエラー・ログを吐きながら開かずの踏切になってしまうのである．

式（5）を導入してしまった原因は，closeという処理とclose状態を混同したことにある．仕様書の書きかたや読みかたによっては起こりうることである．正しくは，「電車がoutするときには，遮断機はclose状態でなければならない」ということである．「遮断機がclose状態であれば電車はoutできる」というとまた別の問題が出てくる．踏み切りは電車に対して通過できないとかいう筋合いのものではない．

仕様の解釈上の問題が明らかになったので，次の問題は「outするときにはclose状態でなければならない」をどのように表現するかというモデル上の問題である．close状態とは，図10のB1ステート・マシンの状態2のことである．したがって，この状態でのみout可能にすれば良い．具体的には，式（2）の代わりに式（6）を使用する．

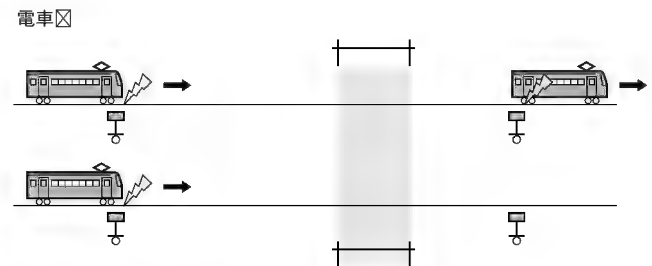


図13　同時イベントの処理

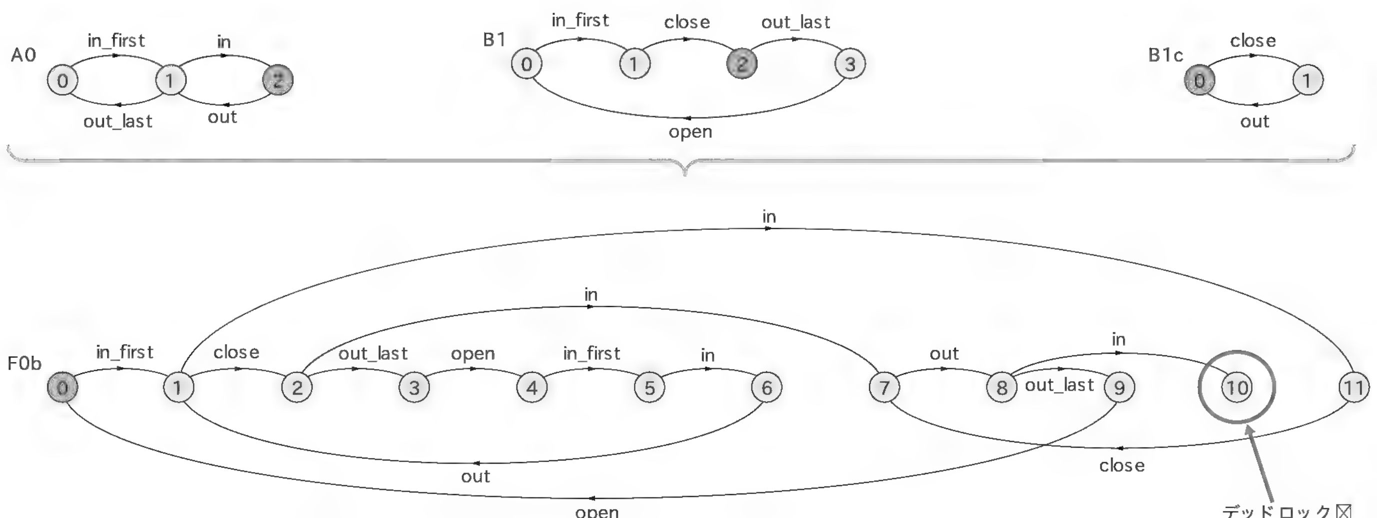


図14　close → outを追加した場合

B1d =(in_first->close->B1f),
B1f =(out->B1f| out_last->open->B1d) ········(6)

この式のB1fローカル状態がclose状態に対応し，この状態でoutメッセージを受理するようになる．この場合のステート・マシンを図15に示す．

B1dステート・マシンの状態2にoutイベントが自己遷移の形で追加されている．また，システム全体のステート・マシンF0cはcloseする前にoutする実行パスもなく，デッドロックすることもない．

式(2)は，制御に関する独立した意味を主張しているので，これは変更せずにこのままにして，別の条件式として，

A0b =(close->A0b1), A0b1=(out->A0b1| open->A0b) ····················(7)

を追加導入しても良い(図16)．

● さらに電車が増えた場合

次に，電車の数をさらに増やした場合どうなるのかを検討してみる．変更になるのは，電車のふるまいを表すA0ステート・マシンである．制御仕様であるB1dはそのまま利用できる．実際に生成したシステムのふるまいは図17のようになる．

図18は図17のF0c5ステート・マシンをRose RealTime用に実装したものである．電車の数は状態で分けずにインスタンス変数countを利用して管理することでcount変数がオーバフローするまで利用できるように拡張してある．また，チョイス・ポイントでoutイベントとout_lastイベントを識別している．この実装のほうが図4で示した対応策よりもコンパクトでパフォーマンスも良い．繰り返しのたびに，ソフトウェアが肥大化していくことを避けることができる．しかも，F0c5ステート・マシンから双模倣性による検証をすることもできる．

以下にこのステート・マシンを生成するためのプロセス式を示す．

```
// 電車 5台
A0b=( in_first->A1)
A1=( in->A2| out_last->A0b)
A2=( in->A3| out->A1)
```

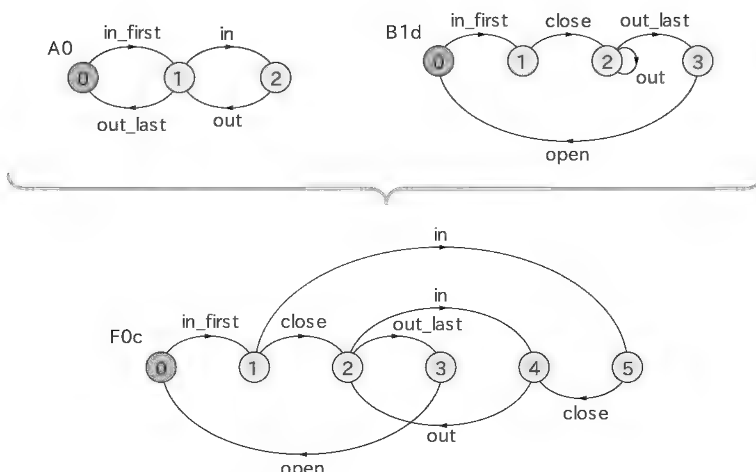


図15　B1dを使用した場合



図16
A0bステート・マシン

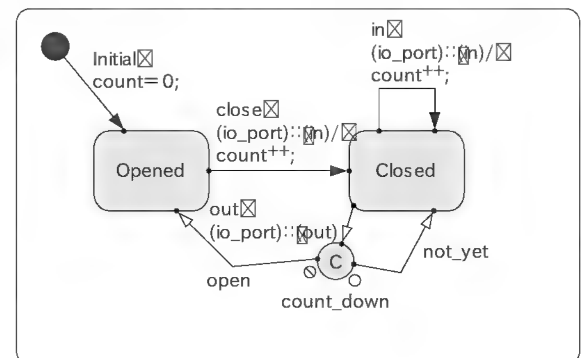


図18　実装例

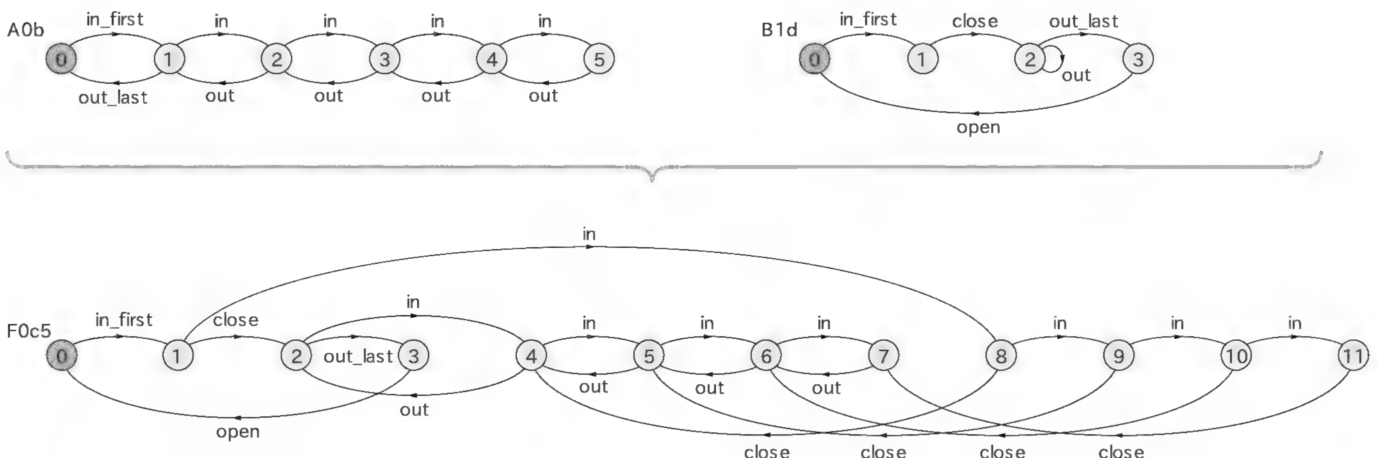


図17　電車が5台の場合

```
A3=(in->A4|out->A2)
A4=(in->A5|out->A3)
A5=(out->A4)

// 制御-2
B1d =(in_first->close->B1f)
B1f =(out->B1f|out_last->open->B1d)

// 踏み切り
||F0c =(A0d||B1d)
```

一つの電車について考えるとinイベントからoutイベントとinイベントからcloseイベントの間には，たとえば，

$$\begin{aligned} t_{close} - t_{in} &\leqq 5sec \\ t_{out} - t_{in} &\geqq 10sec \end{aligned} \quad \cdots\cdots(8)$$

のような時間制約を設定することが可能である．ここで，tは各イベントが発生した時刻を表す．つまり上の式は，電車が接近してから遮断機が下りるまでに5秒以内，電車が接近してから踏み切りを通過するまで10秒以上かかることを示している．この時間制約が遮断機が下りる前に電車が通り過ぎないことを保障する根拠になる．しかし，別の電車が並列して走っている状況でF0c5ステート・マシン上では式(8)の制約は期待できなくなる．inイベント，outイベントが何番目の電車によるものか識別する必要が生じる．この場合には，電車に関するステート・マシンとして，以下のものを使用する．

```
// 電車5台識別
A0c=(in_first->A1)
A1=(in->A2|out_last->A0c)
A2=(in2->A3|out->A1)
A3=(in3->A4|out3->A2)
A4=(in4->A5|out4->A3)
A5=(out5->A4)
```

A0cステート・マシンを使用することで電車ごとのイベントを識別できるようになるので，システム全体のステート・マシンでどの遷移とどの遷移が同時に起こるかがわかるようになる．時間制約については別の機会に説明する．

● デッドロックの例

冒頭の携帯電話の例だけでなく，日本でも読者の多いランボー(Rumbaugh)の本[1]に出てくるATMシステムの例題にもデッドロックのバグがある．いろいろな本や論文で引用されているが，シナリオ・ベースでふるまい合成を行う手法を利用するまで知られていなかったらしい．図19の手順でデッドロックが発生する．手元にランボーの本がある人はトレースしてみてはどうだろうか．

ATM機にカードを入れてパスワードを入力してから，ATM機からのメッセージが返る前にキャンセルを行う．カードを取り出して，再度操作を行う．この二度目の操作のときに口座番号が違っているとデッドロックになる．

displayMainScreen	requestTakeCard
insertCard	takeCard
requestPassword	displayMainScreen
enterPassword	insertCard
verifyAccount	requestPassword
cancel	enterPassword
canceldMessage	verifyCardWithBank
ejectCard	badBankPassword

図19　ATMシステムのデッドロック手順

先の携帯電話の例もそうだが，ここまで深い組み合わせテストはなかなかできない．システムの動作を記述したF0cのようなステート・マシンを作っておかないとテスト空間全体を把握できないので，どこまでテストしたのかがわからない．

F0c5の状態数は12なので簡単にテスト可能だが，ATMの場合は約150状態になる．この程度になるとLTSAのようなツールが必須になる．ツールでデッドロックの有無を調べて仕様レベルでの検証をすることが重要になる．実際の組み込みシステムの状態数は数万以上になる．図14のF0bステート・マシンは電車数がF0c5より少ないのに状態数が12あるように，仕様記述を正しく行わないと実システムでは簡単に状態数が爆発的に増えてしてしまい，ツールも適用できなくなってしまう．この状態爆発とどのように戦うかが形式的手法の課題である．

形式手法自身のアルゴリズムを改善するだけでなく，どのように形式手法を開発プロセスの中に組み込んでいくかもまた重要である．

まとめ

今回は，ハッセ図ではなく直接ステート・マシンを記述して並列性をもったプログラムを構築する方法について説明した．ハッセ図を使えば，アクティブ・オブジェクトの必要数を解析的に決定することができるが，実際のアクティブ・オブジェクトのふるまいを決定するには表現力が少なすぎる．条件分岐が起こるようなふるまいに対してハッセ図は使用できないので，ステート・マシンを使うことになる．ステート・マシンを直接記述するには多少のコツが必要である．プロセス代数式を覚えなければならないが，その代わり仕様記述レベルでの検証が可能になる．特に，デッドロックの回避には有効である．

並列動作する外界と相互作用するシステムでは，バグが発見されたらソース・コードをすぐに直すのではなく，仕様の解釈のレベルから再検討することを薦める．

参考文献
(1) J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

ふじくら・としゆき　(株)豆蔵

MMUなしでも動作するLinux

はじめて使うμClinux

第1回　μClinuxの機能と動作の概要

大谷 浩司

1 はじめに

Linuxは，デマンド・ページング仮想記憶(後述)をサポートするOSです．そのため，通常はMMU(Memory Management Unit)が必須となります．しかし，仮想記憶がサポートされていなくても，Linuxに多くの有用性があることは，想像できるかと思います．たとえば，以下のような利点があります．

1) 多種のファイル・システムをサポート
2) 多種のデバイス・ドライバをサポート
3) 標準でTCP/IPをサポート
4) よく知られたAPI
5) 豊富なライブラリ
6) オープン・ソース

そこで，MMUなしのCPUにLinuxを載せようというプロジェクトが起こりました．それがμClinuxです．μは，「マイクロ」を意味し，「C」はコントローラを意味します．つまり，マイクロコントローラでも動作するLinuxという意味で，「ユーシー」Linuxと発音します．公式Webサイトは，

```
http://www.uclinux.org/
```

です．

最初は，Linuxへのパッチや別ソースの形で提供されていましたが，2.6からは本家のLinuxのソース・ツリーに含まれます．ただし，2.6.4のソースをのぞいてみたところ，いまのところ含まれているのはH8およびM68Kアーキテクチャのものだけのようです．

以下では，簡単な紹介のあと，MMUなしでプログラムが動作するしくみを説明します．その後，それを踏まえて，μClinuxでのプログラムについて解説します．

● デマンド・ページング仮想記憶とは

実際のメモリよりも大きな記憶領域があるかのように見せるしくみが仮想記憶です．実際のメモリの領域を実メモリ，見せかけのメモリの領域を仮想メモリといいます．仮想メモリに与えられたアドレスを仮想アドレス，実メモリのアドレスを実アドレス(または物理アドレス)，仮想アドレスの範囲を仮想アドレス空間(または単に仮想空間)，実アドレスの範囲を実アドレス空間(または単に実空間)と呼びます．

仮想記憶を実現するためには，プログラムのコードやデータのすべてを主記憶上に置かず，一部を補助記憶上に置きます．このためにメモリ空間を比較的小さな固定長の領域(ページ)に分割して管理する方式をページング方式と呼びます．

ページング方式では，実アドレス空間と仮想アドレス空間の対応はページ単位で行います．ページング方式の中でも，プログラムの利用するコードやデータを最初に実メモリ上にロードしないで，必要になった時点でページ単位でロードする方式をデマンド・ページング方式とよびます．

MMUは，このページの実アドレス-仮想アドレス対応を管理し，実アドレス空間に存在しないページをアクセスしようとしたときにCPUに例外を発生させます．これを，ページ・フォールトと呼びます．このため，ページング仮想記憶方式を実装するためには，MMUが必須となります．

2 μClinuxとは

● なぜμClinuxなのか？

前述のように，仮想記憶なしでもLinuxは有用です．しかし，せっかくLinuxを利用するのに，なぜ，わざわざMMUなしのCPUを使うのでしょうか．それにはいくつかの理由が考えられます．

1) コスト

MMUなしのCPUは，MMU付きのCPUよりも安価です．そのため，CPUのコストが問題になる場合には，MMUなしのCPUを選択することがあります．

2) CPUの特徴

MMUはないものの周辺回路や付属のハードウェア，アーキテクチャなどに特徴があり，そのためにそのCPUを使いたい場合があります．

3) 既存ハードウェア

既存のハードウェアをそのまま使いたい場合もあります．実際，μClinuxが最初に動作したのは，PalmPilotのようです．

4) チャレンジ

技術的な興味から，あるCPUでLinuxが動作すればおもし

ろいと思うかもしれません．

このような理由から，μClinuxは，ルータなどの製品のほか，PalmPilotなどでも動作します．また，iPodでも動作するという情報もあります（http://ipodlinux.sourceforge.net/）．

● 動作するハードウェア

現在μClinuxが動作しているCPUには，次のようなものがあります．

1) Motorola DragonBall
2) Motorola ColdFire
3) QUICC
4) ARM7TDMI
5) ADI Blackfin
6) ETRAX
7) Intel i960
8) PRISMA
9) NEC V850E
10) 日立 H8

以上はμClinuxのWebサイトからの情報です．H8用は，佐藤嘉則氏が移植し，メンテナンスを行っています（http://sourceforge.jp/projects/uclinux-h8/）．

11) 富士通 FRV
12) セイコーエプソン S1C33

この二つは（株）アックスが移植し，サポートしています（http://axe-inc.co.jp/）．

動作に必要なリソースは，以下のようなものです．

ARMの場合，苦労せずに動作させるには，以下のメモリがあれば良いでしょう．

- ROM 2Mバイト/RAM 16Mバイト

Webページによると H8の場合は，以下のいずれかのメモリで一応は動作するようです．

- ROM 1Mバイト/RAM 1Mバイト
- RAM 2Mバイト

シリアル・ポートとLANは，付いていたほうがインストールや開発が楽です．個人で試す場合，H8ならば，以下のボードが手に入れやすいでしょう．

- 秋月電子のH8/3068ボード，H8/3609ボード＋増設メモリ
- H8MAX

または，PalmPilotやiPodを買ってμClinuxを入れてみるのも良いかもしれません．

● 通常のLinuxとの違い

仮想記憶サポートがないことと，メモリ容量などの制限からμClinuxは通常のLinuxと次に挙げる点で異なっています．ただし，詳細は各アーキテクチャのポーティングやディストリビューションにより異なるので，利用しようと思っているもので確認する必要があります．

COLUMN

BusyBox

Linuxには，シェルのほかにたくさんの小さいコマンドが存在します．これらを組み合せて複雑なことができるのがUNIX系OSの便利なところです．しかし，一つ一つは小くても，数が多くなればそれなりに大きなサイズになります．しかも，これらは共通で使っている部品も多くあります．それなら，一つのプログラムでこれらを実現すれば，合計のサイズが小さくなるはずです．そうすれば，リソースが厳しいシステムでは，重宝するのではないかということで開発を始められたのがBusyBoxです．

BusyBoxは，シェルのほかに，basename，cat，ls，chgroup，chown，chmod，cmp，cpioなどたくさんのコマンドの機能を含みます．これらの機能は，各コマンド名にリンクすることで，呼び分けるのが通常です．このようにすれば，通常のコマンド群を利用しているのと変わらない利用環境になります．これらの機能は，コンパイル時に必要なものだけを選択可能です．

組み込みに利用する場合は，カーネルに本家のLinuxを使っていてもBusyBoxが使われることも多いようです．

1) APIが異なる

仮想記憶を持たないために，関係するAPIが利用できません．たとえば，forkはサポートされず，vforkのみがサポートされます．

2) libcが異なる

メモリなどの制限からglibcではなく，より小さなuClibcと呼ばれる実装が利用されることが多いです．

3) スタックの扱いが異なる

本家のLinuxでは，プロセスのスタックは自動的に拡張されます．しかし，このしくみは，MMUのメモリ保護機能を利用しているためμClinuxでは利用できません．そのため，あらかじめプログラム・リンク時に指定して，起動時に確保します．

4) オブジェクト形式が異なる

本家のLinuxでは，ELFと呼ばれるオブジェクト形式が利用されていますが．FLATと呼ばれるより小さなオブジェクト形式が使われている場合も多いようです．

5) C++の機能が制限されている

C++の標準ライブラリlibstdc++などは，複雑であるため完全に実装されていない場合も多いようです．そのため，C++でも標準テンプレート・ライブラリなどがそのまま使えません．

6) POSIXスレッドが使えない場合が多い

スレッドで使われるlibpthreadも実装されていない場合が多いようです．

7) 共有ライブラリ機構が使えない

本家の共有ライブラリ機構は，仮想記憶機構を利用している

ために利用できません．ただし，一部のポーティングでは，独自の機構により実装しています．

8) ユーザ・ランドが異なる

これまでに挙げたような理由およびメモリ量などの制限からユーザ・ランドのコマンドは，本家のLinuxとは異なっています．たとえば，シェルなどにBusyBox(コラム参照)が使われる場合があります．

3 μClinuxでのプログラムの動作

ここでは，μClinuxがどのようにMMUなしで複数のプロセスを動作させているかを見てみます．

まず，通常のLinuxのプログラムのアドレス・マップの例を図1に示します．UNIX系のプログラムは，伝統的にテキスト・セグメント，データ・セグメント，BSS，スタックの4個の領域に別れます(現在のLinuxは，実際には，もう少し複雑だが，今回は説明を簡単にするため，ここまでにしておく)．テキスト・セグメントには，プログラムのコードや読み出し専用のデータなどが入ります．データ・セグメントには，明示的に初期化されたデータが入ります．BSSは，明示的に初期化されないデータが入り，0に初期化されます．スタックは，文字どおりの意味に使われます．

図1を見るとわかるように，通常のLinuxでは，プログラムA，プログラムBはともに，プログラム用のアドレス空間のすべてを利用することができ，同じプログラムの同じセグメントは，つねに同じアドレスで始まることが保証されます．スタックは図1では，最上位アドレスから下位アドレスに向って自動的に拡張されます．なお，カーネルに関する領域は，ユーザ・プログラムからは隠されていて見えません．

これを実現するためには，MMUの機能を利用しています．MMUがなければ，各プログラムに独立なアドレス空間を与えることはできません．したがって，同じアドレスで始まることを保証することや，カーネル領域を隠すこともできません．スタックの自動拡張もMMUの保護機能を利用しているため行うことができません．これらの問題に対してμClinuxでは，以下のように対処しています．同一のアドレス空間に複数のプログラムを配置するために，プログラムは異なるアドレスに配置します．カーネルを隠すことはあきらめます．スタックは，固定サイズをあらかじめ割り当てることとし，拡張はあきらめます．

図2にμClinuxでのプログラムのアドレス・マップを示します．図1と異なり，プログラムBがプログラムAの後に配置されています．スタックは各プログラムの最後に固定サイズが割り当られています．カーネルの領域が見えています．この例では，最下位アドレス領域のみが示されていますが，実際にはプログラムとプログラムの間にカーネルが使用する領域が存在することもあります．

プログラム領域の開始アドレスは，そのプログラムの起動時になって初めて決定されます．そのため，μClinuxでは，プログラム起動時にリロケーションを行う必要があります．多くのアーキテクチャでは，リロケーションを高速に行うためにプログラムをいわゆる位置独立コード(PIC)としてコンパイルします．

次に新しいプロセスを作成する場合を考えてみます．Linuxでは，通常は`fork`システム・コールを使います．図3にLinuxで`fork`を実行したときのようすを示します．`fork`では，子プロセスは最初すべてのメモリを共有します．その後，書き換えようとした部分だけが実メモリ上でコピーされ，あらためて同じアドレスにマップされます．これをコピー・オン・ライト(copy on write)動作と呼びます．図3では網かけ部にあたります．

MMUがなければ，コピー・オン・ライト動作が不可能です．そのため，`fork`を実現するためには，実際にコピーしなければいけません．さらに，コピー先はアドレスが異なるため，リロケーションしなければいけません．しかし，プログラムが任意に書き換えたあとのデータ中に含まれるアドレス情報をシステムが安全にリロケーションすることは不可能です．また，`fork`後は，すみやかに`exec`を実行するプログラムが多く，

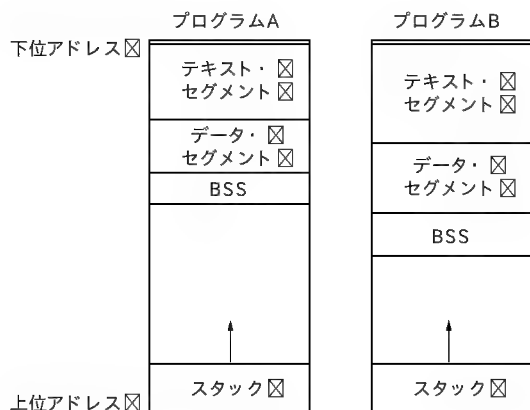


図1 Linuxプログラムのアドレス・マップ

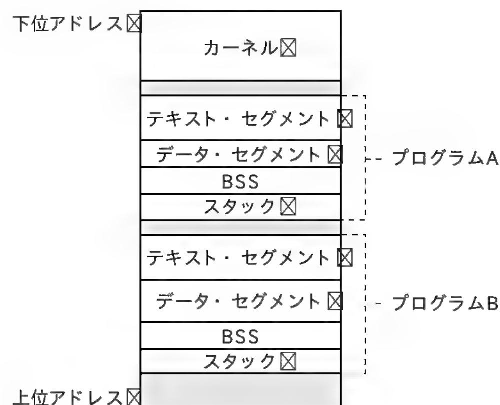


図2 μClinuxプログラムのアドレス・マップ

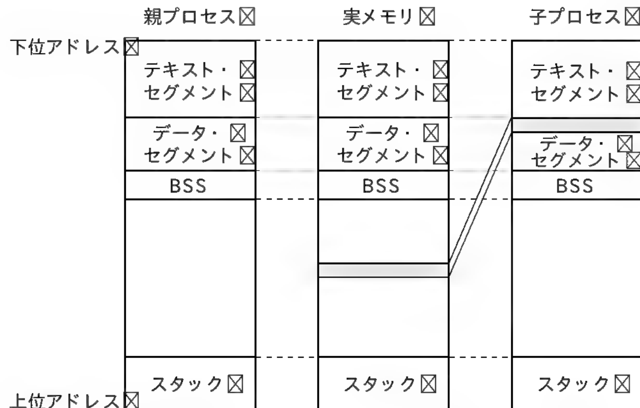


図3 Linuxの`fork`

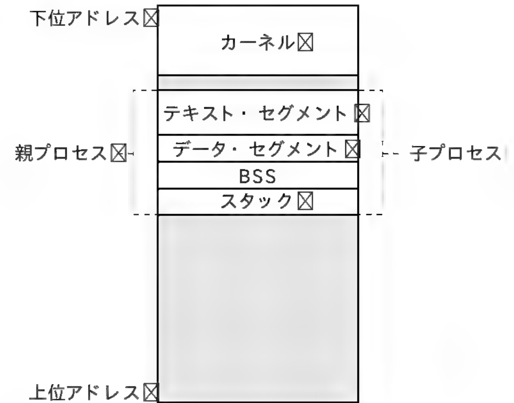


図4 μClinuxの`fork`

コピーは多くの場合にむだになります．そこで，μClinuxでは，forkをサポートすることをあきらめ，プロセスの生成にはvforkを使うことにしています．vforkはforkと異なり，親プロセスは子プロセスが終了するかexec系システム・コールを呼ばない限り，動作を停止しています．また，メモリは親と同じものを使い続けるので，グローバル変数を変更すると，親プロセスにも影響を与えます．図4にμClinuxでvforkを実行したときのようすを示します．

4 μClinuxでのプログラミング

いままで説明してきたように，μClinuxは本家のLinuxと，いくつかの点で異っています．そのため，ちょっと複雑なプログラムでは，本家のLinux向けに書かれたプログラムは，「持ってきてMakeをやり直すだけで移植完了」とはいかないようです．そこで，修正が必要になってきます．ここでは，そのためのプログラムの修正や，新たに書き直す場合の注意点を説明します．

μClinuxでのプログラミングは，基本的には本家のLinuxと同じですが，いくつか気をつけなければならないことがあります．おもなものとして以下の事項があります．

1) forkが使えない
2) メモリがフラグメンテーションを起こす
3) スタックが自動的に拡張されない
4) メモリ空間が保護されていない

これらをもう少し詳細に説明します．

● **fork**が使えない

forkシステム・コールは，サポートされていません．新しいプロセスの生成にはvforkを利用します．

forkのもっとも簡単な例，すなわち指定のプログラム起動だけを行う場合は，forkをvforkに変更するだけです．

たとえば，リスト1のようなコードを考えます．これは，リスト2のように変更します．

リスト1 **fork**を**vfork**に変更する例(変更前)

```
...
cmd = "sample";
arg1 = "-l";
arg2 = "data";
pid = fork();
if (pid == 0) {
    execlp(cmd,cmd,arg1,arg2,NULL);
    _exit(2);
}
...
```

リスト2 **fork**を**vfork**に変更する例(変更後)

```
...
cmd = "sample";
arg1 = "-l";
arg2 = "data";
pid = vfork();
if (pid == 0) {
    execlp(cmd,cmd,arg1,arg2,NULL);
    _exit(2);
}
...
```

しかし，fork後に親子ともに動作を続行するような場合は，もう少しくふうが必要です．たとえば，リスト3のようなコードでは，もう一度自分自身を特別な引き数で起動するようにします．たとえば，リスト4のように変更します．

また，μClinuxでは，一般にテキスト共有やデータ共有はできないので，foo()やbar()の共通部分が少なかったり，foo()やbar()以外の部分も多い場合には，foo()，bar()を異なるプログラムにしたほうがメモリが節約できます．その場合は，リスト5のようになります．

● メモリがフラグメンテーションを起こす

図5に，Linuxの仮想空間と実空間のマッピングの一例を示します．このように，実空間上で連続していなくても仮想空間上では連続したアドレスにマッピングすることができます．そのため，あるサイズの連続した領域が欲しければ，実空間上のあちこちから空いている領域を集めてマッピングすれば良いことになります．

しかし，MMUがない場合には，連続したメモリ領域は，実

リスト 3　fork 後にも動作を続行する例（変更前）

```
...
pid = fork( ) ;
if ( pid == 0)  {
   if ( flag)  {
      foo( ) ;
   } else {
      bar( ) ;
   }
   _exit( 2) ;
}
...
```

リスト 4　fork 後にも動作を続行する例（変更後）

```
int main( int argc, char **argv)
{
   if ( argc > 1)  {
      if ( strcmp( argv[1],"foo")  == 0)  {
         foo( ) ;
      } else if ( strcmp( argv[1],"bar")  == 0)  {
         bar( ) ;
      }
      exit( 2) ;
   }
   通常動作
}
...
pid = vfork( ) ;
if ( pid == 0)  {
   if ( flag)  {
      execlp( "sample","foo",NULL) ;
   } else {
      execlp( "sample","bar",NULL) ;
   }
   _exit( 2) ;
}
...
```

リスト 5　異なるプログラムにした例

```
...
pid = vfork( ) ;
if ( pid == 0)  {
   if ( flag)  {
      execlp( "foo",NULL) ;
   } else {
      execlp( "bar",NULL) ;
   }
   _exit( 2) ;
}
...
```

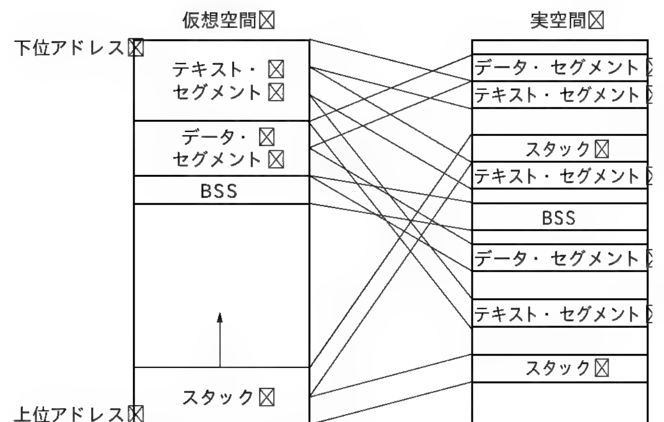


図 5　Linux の仮想空間と実空間

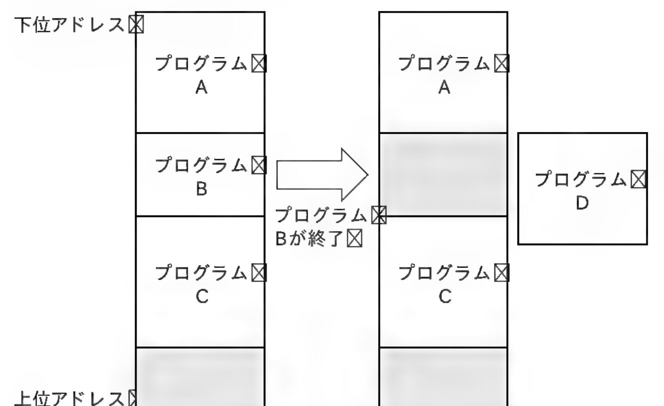


図 6　μClinux のフラグメンテーション

際に連続した領域でなければなりません．図 6 にμClinux のフラグメンテーションを示します．左側では，プログラム A，プログラム B，プログラム C が動作しています．ここで，プログラム B が終了したとします．そうすると，右側のようになります．そこで，プログラム B よりも少々大きいプログラム D を起動しようとしたとします．プログラム B が存在した領域と上位のほうの空き領域を合せるとプログラム D の大きさよりも大きな領域が空いています．しかし，プログラム D は，もちろんプログラム B が存在した場所には入りませんし，上位のほうに余っている領域にも入りません．そのため，起動することはできません．これが，メモリ領域のフラグメンテーション（断片化）です．

このようにプログラムの起動や終了自体でもシステムのメモリ領域のフラグメンテーションを起こします．そのため，大きさの著しく異なるプログラムの終了や起動の繰り返しは避けるようにします．

また，malloc などでカーネルに対して動的に記憶領域を確保，開放要求を繰り返すとシステム全体のメモリ領域がフラグメンテーションを起こします．したがって，カーネルに対してのメモリ領域の要求は，なるべくまとめて行うようにします．

フラグメンテーションは，ひどくなるとシステム全体の停止を招くので注意してください．

● スタックが自動的に拡張されない

スタックは自動的には，拡張されません．あらかじめ割り当てた量を越えないように注意する必要があります．

そのため，配列などの大きなローカル変数の大きさには気をつけてください．構造体のメンバとして配列が定義されているとわかりにくいので注意してください．また，再帰呼出が深くなりすぎないように気をつけなければいけません．

メモリ保護がないため，スタックのオーバフローは，不可解な動作になり，わかりにくいバグとなります．

● メモリ空間が保護されていない

プロセス間でメモリの保護がされていないため，不当なメモリ内容の変更は，当該プロセスだけでなく，ほかのプロセスの動作を不安定にします．それだけでなく，カーネル空間も保護されていないため，システム全体が不安定になり，その後に起動したプログラムにも影響を与えます．メモリ・アクセス関係のバグには，とくに注意しなければなりません．

おおたに・こうじ　（株）アックス

ディップ・スイッチ入力やLED点灯制御，割り込みスイッチに対応した

Linux用PCカード・デバイス・ドライバの作成

山岡 賢一

はじめに

ここでは「TECH I Vol.14 PCカード/メモリカードの徹底研究」で紹介されているPCカードを使った，Linux用PCカード・ドライバの作成手順を紹介します．このハードウェアは，PCMCIA（16ビット），CardBus（32ビット）の両方のPCカード・インターフェースに対応しています．本稿で紹介するドライバも両方のインターフェースに対応します．

なお，本稿で作成したドライバの動作確認には，RedHat 9（Linux 2.4.20）を使用しています．

1 Linux PCカード・ドライバの構造

● PCカードが利用可能になるまでの手順

PCカード・スロットをもつノート・パソコンには，PCカード・コントローラが実装されています．そして，PCカード・コントローラがカードの抜き差しの検出，カード・スロットの電源の制御を行います．PCカードを検出して利用可能になるまでの手順は，次のようになります．

（1）カードの検出

PCカード・コントローラがカード挿入を検出します．PCカード・コントローラからは，PCMCIAかCardBusかなどのカードの種類や，カードの電源電圧を知ることができます．

（2）カードの電源を入れる

PCカード・コントローラを操作して，PCカード・スロットの電源を入れます．

（3）カード情報の取得

PCMCIAカードの場合は，カードのアトリビュート・メモリをCPUのメモリ空間にマップして，CISタプル情報を読み出します．CISタプル情報からは，カードの種別，利用可能なI/O空間，メモリ空間，割り込み情報を知ることができます．CardBusカードの場合は，カードが保持するPCIコンフィグレーション・レジスタを読み出します．通常のPCIデバイスと同様にデバイスの種類を取得できます．

（4）PCカード用レジスタをCPU空間に割り当て

PCカード内に実装されたデバイスのレジスタを操作できるように，CPUのメモリ空間またはI/O空間に割り当てます．PCMCIAの場合は，CISタプルから読み取った情報からI/O空間，メモリ空間を選択して割り当てます．CardBusの場合は，PCIコンフィグレーション・レジスタにカードが使用するI/Oアドレスやメモリ・アドレスを設定します．

（5）デバイス・ドライバ動作

挿入されたPCカード用のデバイス・ドライバは，CPUのメモリ空間またはI/O空間にマップしたレジスタを操作して，デバイスを制御します．

Linuxでは，カードを検出してCPU空間にレジスタを割り当てるまでは，Linuxカーネルが提供する機能を利用します．ここで作成するドライバは「PCカード用デバイス・ドライバ」部分です．

● Linux PCカード用ドライバの構造（PCMCIA）

Linux PCカード用ドライバは，PCMCIAとCardBusでは構造が異なります．それぞれの構造を以下に説明します．

図1にLinuxのPCMCIAドライバの構造を示します．まずはPCMCIAの場合について説明します．

▶ PCカード・コントローラ用ドライバ

PCカード・コントローラLSIの種類ごとに用意されているドライバです．PCカード・コントローラを制御してカードの抜き差し検出，カード・スロットの電源制御，カード情報の読

図1 Linux PCMCIAドライバの構造

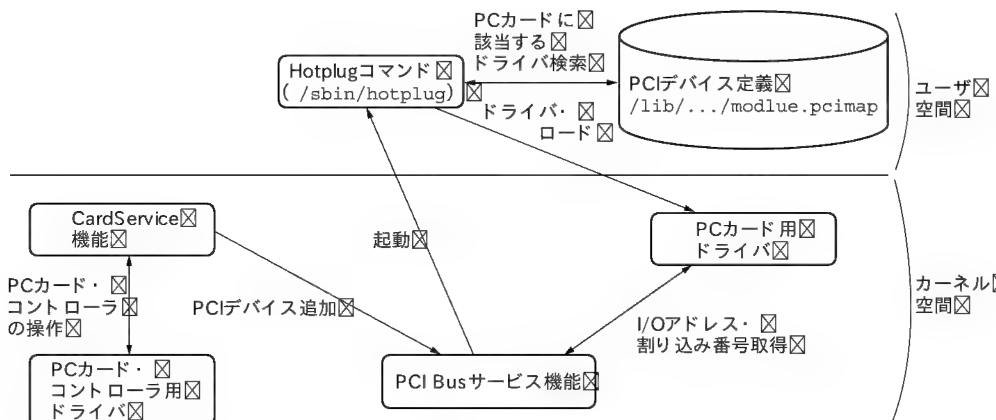


図2 Linux CardBusドライバの構造

み出しを実施します．Linuxカーネル・ソースでは，src/drivers/pcmcia以下に各種PCカード・コントローラ用ドライバが用意されています．

▶ CardService機能

PCカード・コントローラ用ドライバが提供する機能を利用するための関数です．この関数群では，PCカード・コントローラの種類による違いを吸収しており，各カードごとのドライバでは，カード抜き挿しの検出，割り当てたI/O空間，割り込みの取得などで呼び出します．Linuxカーネル・ソースでは，src/drivers/pcmcia/cs.cに実装されています．

▶ PCカード用デーモン・プロセス(cardmgr)

ユーザ空間で動作するデーモン・プロセスです．PCカード・コントローラ用ドライバからカードの抜き挿しのイベントや，カード情報を受け取ります．/etc/pcmciaディレクトリ下の設定ファイルとカード情報を比較して必要なドライバをロードします．

▶ PCカード用ドライバ

PCカードごとに存在するドライバです．CardService関数を呼び出して，I/Oアドレスや割り込み番号を取得します．

● Linux PCカード・ドライバの構造(CardBus)

CardBusカードは，Linuxカーネル2.4から実装されているhotplug機能を利用しています．この機能により，PCI，USB，IEEE1394，ISAバスのPnPデバイスを検出したときに自動的にドライバをロードします．CardBusタイプのPCカードはPCIデバイスのhotplug機能により，PCカード用ドライバをロードして動作します．

図2にLinuxのCardBusドライバの構造を示します．

▶ PCカード・コントローラ用ドライバ

PCカード・コントローラ用ドライバは，PCMCIAと共通です．PCカード・コントローラがCardBus対応であれば，CardBusカードの抜き差し検出，電源管理にも対応します．Linuxカーネル・ソースでは，drivers/pcmcia/cardbus.cにCardBus関連の機能が実装されています．

▶ CardService機能

CardService機能では，CardBusタイプのカード挿入を検出すると，PCI Busサービス機能のPCIデバイス追加を呼び出します．

▶ PCI Busサービス機能

PCI Busサービス機能は，Linuxカーネルが提供するPCIデバイスへのアクセス関数群です．CardService機能からPCIデバイスの追加が呼び出されるとhotplugコマンドを実行します．PCIデバイスのベンダ番号，デバイス番号などは，hotplugコマンドの引き数に与えられます．PCI Busサービス機能は，Linuxソースのdrivers/pci/pci.cに実装されています．

▶ hotplugコマンド

hotplugコマンドは，PCIデバイスの追加によって呼び出されると下記のPCIデバイス定義ファイルから該当するドライバを検索して，PCカード用ドライバをLinuxカーネルにロードします．

/lib/modules/<カーネル・バージョン>/modules.pcimap

▶ PCカード用ドライバ

hotplugコマンドによってロードされたPCカード用ドライバは，PCI Busサービス機能を呼び出して，I/Oアドレスや割り込み番号を取得して動作します．

2 ドライバの作成

● ハードウェアの構成

デバイス・ドライバを作成するターゲット・ハードウェアは，「TECH I Vol.14 PCカード/メモリカードの徹底研究」において紹介されているPCカード(写真1)です．PCカード上にはPLDが実装されており，PLD用シリアルROMを交換することによって，PCMCIA仕様カード，CardBus仕様カードの切り替えが可能です．

カードの拡張部分にはディップスイッチ，LED，割り込みボタンが実装されています．ディップスイッチ設定は，PIOからの読み出しによって取得できます．LEDは，PIOへの出力と連

表1　PCカードの仕様

	PCMCIA		CardBus	
カード検出情報	タプル製品情報 Kurusugawa-ele. PIO PCCard		ベンダID　= 0x6809 デバイスID = 0x8200	
割り込み	1本(非共有)		1本(共有される可能性有り)	
メモリ	なし		連続1Mバイトのメモリ・ウィンドウを占有	
I/Oポート	連続した8バイトを使用		連続した8バイトを使用	
I/Oレジスタ構成(CardBus & PCMCIA共通)	オフセット	ビット長	名　称	動　作
	+0	16	I/Oレジスタ	Read : ディップスイッチ入力 Write : LED出力
	+2	16		未使用
	+4	16	割り込みステータス (ビット0のみ)	Read : 割り込みステータス Write : 割り込み要求クリア
	+6	16	割り込みマスク (ビット0のみ)	Read : 割り込みマスク状態取得 Write : 割り込みマスク設定
	●割り込みステータス ― 割り込み発生でビット0が1に設定される ●割り込み要求クリア ― 1を書き込むと割り込みをクリア ●割り込みマスク ――― 1で割り込み許可，0で割り込み禁止			
メモリ構成	なし		先頭16バイトのみRAMとして使用可能	

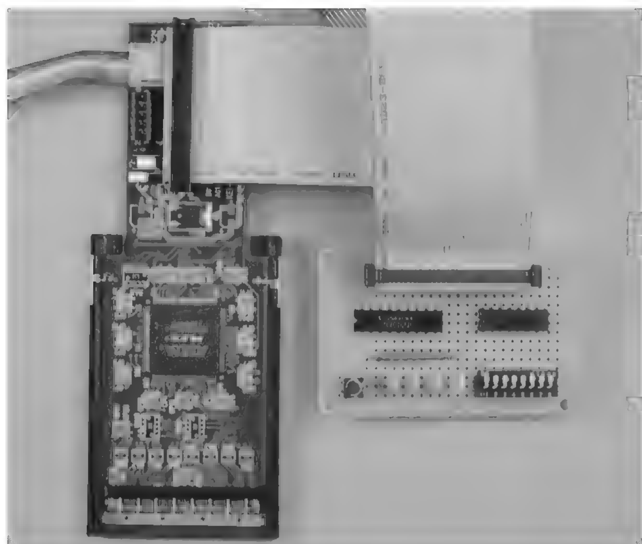


写真1　PCカードの外観

表2　ioctlコマンド一覧

IOCTLコマンド	内　容
CQPIO_SET_LED	引き数にint型を指定. 引き数指定値をLEDに出力する
CQPIO_GET_DIPSW	引き数int型ポインタを指定. 引き数にディップスイッチのON/OFFを読み出す
CQPIO_WAIT_INTERRUPT	割り込みボタン押下まで待つ. ioctlシステム・コールがブロックし，ボタン押下でブロックが解除する

リスト1　ドライバ使用例(sample.c)

```
#include "cqpio.h"
void sample(void)
{
    int fd, sw;
    /* デバイス・ドライバのオープン */
    fd = open("/dev/cqpio0",O_RDONLY);
    /* ディップスイッチ設定読み出し */
    ioctl(fd, CQPIO_GET_DIPSW, &sw);
    /* LED 設定(5=0101) */
    ioctl(fd, CQPIO_SET_LED, 5);
    /* 割り込みボタン発生押下待ち */
    ioctl(fd, CQPIO_WAIT_INTERRUPT, 0);
    /* デバイス・クローズ */
    close(fd);
}
```

動して点灯します．割り込みボタンは押下によってPCカードから割り込みを発生させることができます．表1にPCカードの仕様を示します．

本稿で作成するドライバはCardBus，PCMCIAドライバのレジスタ制御部分を共通化しています．CardBusカードにのみメモリ空間に16バイトのレジスタが存在しますが，今回この部分は使用していません．

● ドライバの仕様

ドライバでは，ディップスイッチの読み出し，LEDへの書き込み，割り込みボタン押下検出の機能を提供します．すべての機能はioctlシステム・コールに割り当てて実装しています(表2)．アプリケーションからのデバイス・ドライバの使用例をリスト1に示します．

● ドライバの構成

ドライバは，PCMCIAとCardBusの両方に対応します．PCMCIAとCardBusではPIOを操作するI/Oレジスタの先頭アドレスが異なる場合がある(ほとんどの場合そうなる)以外は，レジスタの構成は同じです．PIO操作を行うドライバ本体を1か所にまとめ，PCMCIA，CardBusそれぞれの依存部分を独立したモジュールとして実装します(図3)．

cqpio.oはPIO操作デバイス・ドライバ本体です．デバイス・ファイルを経由してアプリケーションからioctl要求を受け取り，処理します．

cqpio_cs.oはPCMCIA依存部です．Linuxカーネルのカード・サービス機能とやりとりしてカードを設定し，I/Oアドレスと割り込み番号をドライバ本体に渡します．

cqpio_pci.oはCardBus依存部です．カードのPCIコン

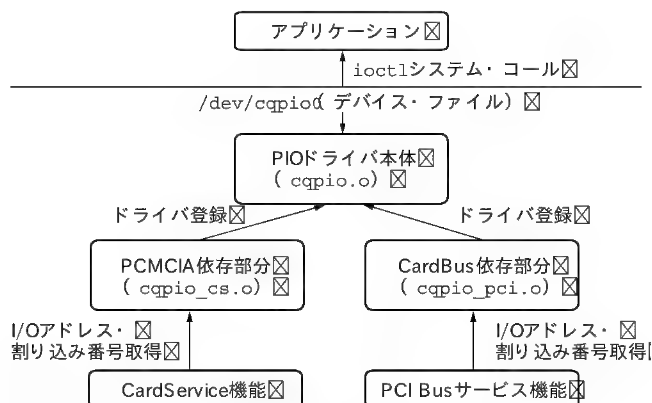


図3 ドライバ・モジュール構成

フィグレーション・レジスタからI/Oアドレス，割り込み番号を読み出してドライバ本体に渡します．

● PIO操作デバイス・ドライバ本体

リスト2のcqpio.cがPIO操作デバイス・ドライバ部分のソースコードです．

cqpio.cにはPCMCIA依存部，CardBus依存部からの呼び出しインターフェースとして，ドライバを登録するregister_cqpio関数(リスト2のⒶ)とドライバを削除するunregister_cqpio関数(同Ⓑ)を用意しています．register_cqpio関数の引き数には，PIO操作用のレジスタのI/Oアドレスと割り込み番号を渡します．このI/Oアドレスと割り込み番号を使用して，デバイス・ドライバとして動作するように設定します．

resiter_cqpio，unregister_cqpio関数はPCMCIA依存部，CardBus依存部から呼び出せるようにリスト2のⒸの部分でEXPORT_SYMBOLによって外部からの呼び出しを可能

リスト2 PIO操作デバイス・ドライバ本体(cqpio.c)

```
/* List.2: PIOドライバ本体(cqpio.c) */
#include <linux/config.h>
#include <linux/module.h>
#include <linux/ioport.h>
#include <linux/kernel.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include "cqpio.h"

/* デバイスごとのデータを管理するテーブル */
static cqpio_dev_data_t g_cqpio_dev_tbl[CQPIO_DEV_MAXCNT];

/*========================================*/
/* 割り込みハンドラ                         */
/*========================================*/
static void cqpio_interrupt(int irq, void *dev_instance,
                                      struct pt_regs *regs)
{
    ushort              status;
    cqpio_dev_data_t* pdata;

    pdata = (cqpio_dev_data_t*)dev_instance;

    /* 割り込みステータス読出し */
    status = inw(pdata->iobase+4);
    if (status & 0x0001) {
        /* 割り込みクリア */
        outw(1, pdata->iobase+4);
        if (pdata->enable && pdata->isopen) {
            /* 割り込み発生待ちプロセスの休眠解除 */
            wake_up_interruptible(&pdata->waitq);
        }
    }
}                                                     ←Ⓕ
/*================================================*/
/* ドライバ・エントリ関数                            */
/*================================================*/
/* OPEN 関数 */
static int cqpio_open(struct inode *inode, struct file *file)
{
    unsigned int       minor;
    cqpio_dev_data_t *pdata;
    minor = MINOR(file->f_dentry->d_inode->i_rdev);
    /* マイナ番号用のデバイス・データ */
    pdata = &g_cqpio_dev_tbl[minor];
    if (pdata->enable == 0) {
        /* デバイスが接続されていない */
        return -ENODEV;
    }
    if (pdata->isopen) {
        /* すでにオープン済み */
        return -EBUSY;
    }
    /* 多重オープン禁止フラグ ON */
    pdata->isopen = 1;
    /* プライベート・データに保存 */
    file->private_data = pdata;
    /* 割り込み許可 */
    outw(1, pdata->iobase+6);
    MOD_INC_USE_COUNT;
    return 0;
}
/* CLOSE 関数 */
cqpio_close(struct inode *inode, struct file *file)
{
    cqpio_dev_data_t* pdata;

    pdata = (cqpio_dev_data_t*)file->private_data;
    if (pdata == NULL || pdata->enable == 0)
        return -ENODEV;

    if (pdata->isopen) {
        /* 割り込み禁止 */
        outw(0, pdata->iobase+6);
        pdata->isopen = 0;
    }

    MOD_DEC_USE_COUNT;
    return 0;
}
/* IOCTL 関数 */
static int cqpio_ioctl(struct inode *inode, struct file
                *file, unsigned int cmd, unsigned long arg)
{
    int led, dip;
    cqpio_dev_data_t* pdata;
    pdata = (cqpio_dev_data_t*)file->private_data;
    switch(cmd) {
    case CQPIO_SET_LED:
        /* LED 設定 */
        outw((unsigned short)arg, pdata->iobase);
        break;
    case CQPIO_GET_DIPSW:
        /* ディップスイッチ読み出し */
        dip = inw(pdata->iobase);
         /*ユーザ空間へコピー*/
        if (copy_to_user((int*)arg, &dip, sizeof(dip)))      ←Ⓔ
            return -EFAULT;
        break;
    case CQPIO_WAIT_INTERRUPT:
        /* 割り込みボタン押下 INTERRUPT 待ち */
        interruptible_sleep_on(&pdata->waitq);
        if (signal_pending(current)) {
            /* シグナル中断 */
```

にしてあります．

ドライバの機能は，ドライバのロード時の初期化関数cqpio_init_module関数（リスト2のⒹ）でキャラクタ型ドライバとして登録しています．アプリケーションのioctlシステム・コールによって，cqpio_ioctl関数（同Ⓔ）が呼び出され，PIOレジスタを操作します．

割り込みボタン待ちのioctlは，cqpio_ioctl関数のCQPIO_WAIT_INTERRUPT要求処理でinterrupt_sleep_on関数で割り込みを待ちます．割り込みハンドラcqpio_interrupt関数（同Ⓕ）でwake_up_interrupt関数を呼び出して待ちを解除しています．

● PCMCIA依存部分の作成

リスト3のcqpio_cs.cがLinuxカーネルのCardService機能を利用するPCMCIA依存部分です．

モジュール・ロード時の初期化init_cqpio_cs関数（リスト3のⒶ）では，register_pccard_driver関数により，該当カード組み込み時の呼び出しcqpio_attch関数と終了処理cqpio_detach関数を登録しています．

cqpio_attch関数（同Ⓑ）は，CardService機能によりドライバ・ロード時に呼び出されます．ここでは，ドライバが対応するカード・タイプ条件を設定してCardService機能への登録を行います．登録時にカード・リリース処理としてcqpio_cs_release関数，カード・イベント・コールバックとしてcqpio_cs_event関数を設定しています．

cqpio_cs_event関数（同Ⓒ）は，カードの抜き差しなどのイベントにより呼び出されるコールバック関数です．カード挿入イベント（CS_EVENT_CARD_INSERTION）によりカード・コンフィグレーション処理関数cqpio_configを呼び出します．

cqpio_config関数（同Ⓓ）は，CardService機能により解析されたCISタプル情報を取得します．取得結果からI/Oアド

リスト2　PIO操作デバイス・ドライバ本体（cqpio.c）（つづき）

```
            return -EINTR;
        }
        if (! pdata->enable) {
            /* WAIT中にデバイスが抜かれた */
            return -ENODEV;
        }
        break;
    default:
        return -ENOIOCTLCMD;
    }
    return 0;
}
/*================================================*/
/* モジュール組み込み・削除処理                   */
/*================================================*/
/* filespec 構造体(ドライバ登録用) */
static struct file_operations cqpio_fileopes = {
    owner:          THIS_MODULE,
    ioctl:          cqpio_ioctl,
    open:           cqpio_open,
    release:        cqpio_close,
};
/* モジュール・ロード関数 */
static int __init cqpio_init_module(void)
{
    int res;
    /* キャラクタ型ドライバ登録 */                    ←Ⓓ
    res = register_chrdev(CQPIO_MAJOR, "cqpio",
&cqpio_fileopes);
    return res;
}
/* モジュール削除関数 */
static void __exit cqpio_cleanup_module(void)
{
    /* ドライバ削除 */
    unregister_chrdev(CQPIO_MAJOR, "cqpio");
}
/* モジュール用定義 */
module_init(cqpio_init_module);
module_exit(cqpio_cleanup_module);
/*=====================================*/
/* PIOドライバ登録・削除関数           */
/*=====================================*/
/*  */
cqpio_dev_data_t* register_cqpio(unsigned long iobase,
                                  int irq)
{
    int               minor, result;
    cqpio_dev_data_t* pdata = NULL;                  ←Ⓐ
    for(minor = 0; minor < CQPIO_DEV_MAXCNT; minor++) {
        if (g_cqpio_dev_tbl[minor].enable == 0) {
            /* デバイス有効フラグ ON */
            pdata = &g_cqpio_dev_tbl[minor];
            pdata->enable = 1;
            break;
        }
    }
    if (pdata == NULL) {
        printk(KERN_ERR "cqpio: register_cqpio():
                         no more register.¥n");
        return NULL;
    }
    /* 割り込み禁止 */
    outw(0, pdata->iobase+6);
    /* 割り込みハンドラ登録 */
    result = request_irq(irq, cqpio_interrupt, SA_SHIRQ |
                         SA_INTERRUPT,
                         "cqpio", (void*)pdata);
    if (result < 0) {
        release_region(pdata->iobase, 8);
        goto error;
    }
    pdata->iobase = iobase;
    pdata->irq    = irq;
    pdata->minor  = minor;
    /* インタラプト用 wait キュー初期化 */
    init_waitqueue_head(&pdata->waitq);
    return pdata;

error:
    pdata->enable = 0;
    return NULL;
}
/* PIOドライバ削除*/
void unregister_cqpio(cqpio_dev_data_t* pdata)
{
    if (pdata == NULL)
        return;
    if (pdata->enable == 1) {
        /* 割り込み禁止 */
        outw(0, pdata->iobase+6);                    ←Ⓑ
        /* デバイス有効フラグ OFF  */
        pdata->enable = 0;
        /* 割り込みハンドラ解除 */
        free_irq(pdata->irq, pdata);
    }
}
/* 外部モジュールから呼び出すため */
EXPORT_SYMBOL (register_cqpio);                      ←Ⓒ
EXPORT_SYMBOL (unregister_cqpio);
```

リスト 3　PCMCIA 依存部分(cqpio_cs.c)

```
/* List.3: cqpio_cs.c PCMCIA 依存部分 */
#include <linux/kernel.h>
#include <linux/version.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/ptrace.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/timer.h>
#include <linux/ioport.h>
#include <pcmcia/version.h>
#include <pcmcia/cs_types.h>
#include <pcmcia/cs.h>
#include <pcmcia/cistpl.h>
#include <pcmcia/ds.h>
#include <pcmcia/cisreg.h>
#include <pcmcia/ciscode.h>
#include "cqpio.h"
static void cqpio_cs_release(u_long arg);
static void cqpio_detach(dev_link_t *link);
/*====================================================*/
typedef struct cqpio_info_t {
    dev_link_t          link;
    int                 ndev;
    dev_node_t          node;
    cqpio_dev_data_t    *dev_data;
} cqpio_info_t;

static dev_info_t  dev_info = "cqpio_cs";
static dev_link_t *dev_list = NULL;

/* カード・サービスへのエラー通知関数 */
static void cs_error(client_handle_t handle, int func, int ret)
{
    error_info_t err = { func, ret };
    CardServices(ReportError, handle, &err);
}
#define CS_CHECK(fn, args...) ¥
  if ((last_ret=CardServices(last_fn=(fn), args))!=0) goto
                                                     cs_failed
#define CFG_CHECK(fn, args...) ¥
  if (CardServices(fn, args) != 0) goto next_entry

/*====================================================*/
/* カード・コンフィグレーション処理                    */
/*====================================================*/
static void cqpio_config(dev_link_t *link)
{
    client_handle_t handle = link->handle;
    cqpio_info_t *info = link->priv;
    tuple_t tuple;
    u_short buf[128];
    cisparse_t parse;
    config_info_t conf;
    cistpl_cftable_entry_t *cfg = &parse.cftable_entry;
    cistpl_cftable_entry_t dflt = { 0 };
    cqpio_dev_data_t *p;
    int i, last_ret, last_fn;

    tuple.TupleData     = (cisdata_t *)buf;
    tuple.TupleOffset   = 0; tuple.TupleDataMax = 255;
    tuple.Attributes    = 0;
    tuple.DesiredTuple = CISTPL_CONFIG;
    /* 先頭のタプルを取出し */
    CS_CHECK(GetFirstTuple, handle, &tuple);
    CS_CHECK(GetTupleData, handle, &tuple);
    CS_CHECK(ParseTuple, handle, &tuple, &parse);
    link->conf.ConfigBase = parse.config.base;
    link->conf.Present     = parse.config.rmask[0];

    /* カード・コンフィグレーション中 */
    link->state |= DEV_CONFIG;
    CS_CHECK(GetConfigurationInfo, handle, &conf);
    tuple.DesiredTuple = CISTPL_CFTABLE_ENTRY;
    tuple.Attributes = 0;
    /* 先頭のタプルを取出し */
    CS_CHECK(GetFirstTuple, handle, &tuple);
    /* タプルを順次参照して使用するものを決める */
    while(1) {
        /* タプル・データを解析 */
        CFG_CHECK(GetTupleData, handle, &tuple);
        CFG_CHECK(ParseTuple, handle, &tuple, &parse);
        if ((cfg->io.nwin > 0) || (dflt.io.nwin > 0)) {
            cistpl_io_t *io = (cfg->io.nwin) ? &cfg
                                 ->io : &dflt.io;
            link->conf.ConfigIndex = cfg->index;
            link->io.BasePort1     = io->win[0].base;
            link->io.NumPorts1     = io->win[0].len;
            link->io.IOAddrLines   = io->flags &
                                       CISTPL_IO_LINES_MASK;
            CFG_CHECK(RequestIO, link->handle, &link->io);
            break;
        }
    next_entry:
        /* 次のタプルを参照する */
        if (cfg->flags & CISTPL_CFTABLE_DEFAULT) dflt = *cfg;
        CS_CHECK(GetNextTuple, handle, &tuple);
    }
    CS_CHECK(RequestIRQ, handle, &link->irq);
    CS_CHECK(RequestConfiguration, handle, &link->conf);
    /* ★PIO ドライバの登録(cqpio.o 呼び出し) */
    p = register_cqpio(link->io.BasePort1,
                                 link->irq.AssignedIRQ);
    if (p == NULL) {
        goto failed;
    }
    info->ndev       = 1;
    info->node.major = CQPIO_MAJOR;
    info->node.minor = p->minor;
    info->dev_data   = p;
    strcpy(info->node.dev_name, "cqpio");
    link->dev        = &info->node;
    /* コンフィグレーションの完了 */
    link->state &= ~DEV_CONFIG_PENDING;
    return;
cs_failed:
    cs_error(link->handle, last_fn, last_ret);
failed:
    cqpio_cs_release((u_long)link);
}
/*=============================================*/        ┐
/* カード・リリース処理                        */        │
/*=============================================*/        │
static void cqpio_cs_release(u_long arg)                 │
{                                                        │
    dev_link_t   *link = (dev_link_t *)arg;              │
    cqpio_info_t *info = link->priv;                     │
    if (info->ndev) {                                    │
        /* PIO ドライバ削除(cqpio.c 内の関数呼び出し) */  ├─Ⓓ
        unregister_cqpio(info->dev_data);                │
    }                                                    │
    info->ndev = 0;                                      │
    link->dev  = NULL;                                   │
    /* コンフィグレーション情報の開放処理 */             │
    CardServices(ReleaseConfiguration, link->handle);    │
    CardServices(ReleaseIO, link->handle, &link->io);    │
    CardServices(ReleaseIRQ, link->handle, &link->irq);  │
    link->state &= ~DEV_CONFIG;                          │
                                                         │
}                                                        ┘
/*================================================*/
/* カード・サービス・イベント処理関数              */
/*================================================*/     ┐
static int cqpio_cs_event(event_t event, int priority,   │
                        event_callback_args_t * args)    │
{   dev_link_t   *link = args->client_data;              │
    switch (event) {                                     │
    case CS_EVENT_CARD_INSERTION: /* カード挿入 */       │
        /* コンフィグレーション開始 */                   │
        link->state |= DEV_PRESENT | DEV_CONFIG_PENDING; │
        /* コンフィグレーション処理(List.3-1)呼び出し */ │
        cqpio_config(link);                              │
        break;                                           │
    case CS_EVENT_CARD_REMOVAL: /* カードが抜かれた */   ├─Ⓒ
        link->state &= ~DEV_PRESENT;                     │
        if (link->state & DEV_CONFIG) {                  │
            /* カード・リリース処理をタイマで呼び出し設定 */│
            mod_timer(&link->release, jiffies + HZ/20);  │
        }                                                │
        break;                                           │
    case CS_EVENT_PM_SUSPEND: /* サスペンド */           │
        link->state |= DEV_SUSPEND;                      │
        /* no break */                                   │
```

リスト 3　PCMCIA 依存部分(cqpio_cs.c)(つづき)

```
    case CS_EVENT_RESET_PHYSICAL: /* カード・リセット */
        if (link->state & DEV_CONFIG) {
            CardServices(ReleaseConfiguration, link->handle);
        }
        break;
    case CS_EVENT_PM_RESUME: /* レジューム */
        link->state &= ~DEV_SUSPEND;
        /* no break */
    case CS_EVENT_CARD_RESET: /* カード・リセット終了 */
        if (DEV_OK(link)) {
            CardServices(RequestConfiguration,
                                 link->handle, &link->conf);
        }
        break;
    }
    return 0;
}
/*==========================================*/
/*:CardService機能への登録・削除              */
/*==========================================*/
/* CardService 機能への登録 */
static dev_link_t *cqpio_attach(void)
{
    cqpio_info_t *info;
    dev_link_t   *link;
    client_reg_t  client_reg;
    int           i, ret;
    /* ドライバ・データ用領域確保 */
    info = kmalloc(sizeof(*info), GFP_KERNEL);
    if (!info) return NULL;
    memset(info, 0, sizeof(*info));
    link = &info->link; link->priv = info;
    /* カード・リリース処理関数(List.3-2)登録 */
    link->release.function = &cqpio_cs_release;
    link->release.data     = (u_long)link;
    /* ドライバが対応するカード・タイプ条件 */
    link->io.Attributes1   = IO_DATA_PATH_WIDTH_16;
    link->io.Attributes2   = 0;
    link->irq.Attributes   = IRQ_TYPE_EXCLUSIVE;
    link->irq.IRQInfo1     = IRQ_INFO2_VALID|IRQ_LEVEL_ID;
    link->irq.IRQInfo2     = 0xdeb8;
    link->conf.Attributes  = CONF_ENABLE_IRQ;
    link->conf.Vcc         = 50;
    link->conf.IntType     = INT_MEMORY_AND_IO;
    /* カード・サービスへの登録 */
    link->next = dev_list;
    dev_list   = link;
    client_reg.dev_info   = &dev_info;
    client_reg.Attributes = INFO_IO_CLIENT | INFO_CARD_SHARE;
    /* イベント処理関数呼び出し条件マスク設定 */
    client_reg.EventMask  =
        CS_EVENT_CARD_INSERTION | CS_EVENT_CARD_REMOVAL |
        CS_EVENT_RESET_PHYSICAL | CS_EVENT_CARD_RESET |
        CS_EVENT_PM_SUSPEND     | CS_EVENT_PM_RESUME;
    /* カード・イベント処理関数(List.3-3)登録 */
    client_reg.event_handler = &cqpio_cs_event;

    client_reg.Version       = 0x0210;
    client_reg.event_callback_args.client_data = link;
    ret = CardServices(RegisterClient,
                              &link->handle, &client_reg);
    if (ret != CS_SUCCESS) {
        cs_error(link->handle, RegisterClient, ret);
        cqpio_detach(link);
        return NULL;
    }
    return link;
}
/* CardService機能からの削除 */
static void cqpio_detach(dev_link_t *link)
{
    dev_link_t **linkp;
    int ret;
    for (linkp = &dev_list; *linkp; linkp = &(*linkp)->next) {
        if (*linkp == link)
            break;
    }
    if (*linkp == NULL)
        return;
    del_timer(&link->release);
    if (link->state & DEV_CONFIG) {
        cqpio_cs_release((u_long)link);
    }
    if (link->handle) {
        ret = CardServices(DeregisterClient, link->handle);
        if (ret != CS_SUCCESS) {
            cs_error(link->handle, DeregisterClient, ret);
        }
    }
    *linkp = link->next;
    kfree(link->priv);
}
/*=======================================*/
/* モジュール登録・削除処理                 */
/*=======================================*/
/* モジュール登録時の呼び出し関数 */
static int __init init_cqpio_cs(void)
{
    /* PCMCIA ドライバへの登録                */
    /*  cqpio_attach, cqpio_detach 関数を登録 */
    register_pccard_driver(&dev_info, &cqpio_attach,
&cqpio_detach);
    return 0;
}
/* モジュール削除時の呼び出し関数 */
static void __exit exit_cqpio_cs(void)
{
    /* PCカード・ドライバからの削除 */
    unregister_pccard_driver(&dev_info);
    while (dev_list != NULL) {
        cqpio_detach(dev_list);
    }
}
/* モジュール登録・削除エントリ定義 */
module_init(init_cqpio_cs);
module_exit(exit_cqpio_cs);
```

レス，割り込み番号を引き数にしてPIO操作ドライバ本体のregister_cqpio関数を呼び出し，ドライバを使用可能にします．

● CardBus 依存部分の作成

リスト4のcqpio_pci.cが，Linuxカーネルのhotplug機能を利用してロードするCardBus依存部分です．CardBus用ドライバはPCIデバイス用ドライバとして記述します．

モジュール・ロード時の初期化関数init_module(リスト4のⒶ)ではpci_module_init関数を呼び出してPCIドライバとしての登録を実施しています．この登録によってリスト4のⒷのテーブルに定義したPCIデバイス検出により，cqpio_probe関数を呼び出すようになります．cqpio_probe関数(リスト4のⒸ)ではPCIコンフィグレーション・レジスタを読み出して，I/Oアドレスと割り込み番号を取得し，PIO操作ドライバ本体のregister_cqpio関数を呼び出します．

3 動作確認

● コンパイルとインストール

(1) コンパイル

リスト5に示すMakefileを使用してコンパイルを実施します．次のようにmakeを実行すると，cqpio.o，cqpio_pci.o，

cqpio_cs.oの三つのモジュールを生成します．

```
% make
```

(2) モジュール・ファイルのコピー

/lib/modules/<バージョン>/cqpioディレクトリに三つのモジュールをコピーします．スーパ・ユーザ権限で次のコマンドを実行します．

```
# make install
```

(3) モジュール情報の更新

今回作成したドライバでは「モジュール間の呼び出し依存関係」，「PCIデバイス情報」の二つのモジュール情報を更新する必要があります．これらの情報を更新するためにスーパ・ユーザ権限で以下のコマンドを実行します．

```
# /sbin/depmod -a
```

モジュール間の呼び出し依存関係は，modules.depファイ

リスト4　CardBus依存部分(cqpio_pci.c)

```
/* List.4: cqpio_pci.c: CardBus依存部分 */
#include <linux/modversions.h>
#include <linux/config.h>
#include <linux/version.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/string.h>
#include <linux/errno.h>
#include <linux/slab.h>
#include <linux/pci.h>
#include "cqpio.h"
/*========================================*/
/* PCI ドライバ用 PROBE 関数              */
/*========================================*/
static int __devinit cqpio_probe(                          ┐
    struct pci_dev             *dev,                       │
    const struct pci_device_id *pci_id                     │
)                                                          │
{                                                          │
    unsigned long    iobase;                               │
    int              irq;                                  │
    cqpio_dev_data_t *pdata;                               │
                                                           │
    if (pci_enable_device(dev))                            │
        return -ENODEV;                                    │
    /* I/O, メモリの割り当て */                            │
    if (pci_request_regions(dev,"cqpio")) {                │
        pci_disable_device(dev);                           │
        return -ENODEV;                                    ├─Ⓒ
    }                                                      │
    /* I/O 空間開始アドレス取得 */                         │
    iobase = pci_resource_start(dev, 0);                   │
    irq    = dev->irq;                                     │
    /* cqpio ドライバ登録(戻り値はドライバデータ) */       │
    pdata = register_cqpio(iobase, irq);                   │
    if (pdata == NULL) {                                   │
        pci_release_regions(dev);                          │
        pci_disable_device(dev);                           │
        return -ENODEV;                                    │
    }                                                      │
    /* ドライバデータ保存 */                               │
    pci_set_drvdata(dev,pdata);                            │
    return 0;                                              │
}                                                          ┘
/*============================================*/
/* PCIデバイス削除                            */
/*============================================*/
static void __devexit cqpio_remove(
    struct pci_dev *dev
)
{
    cqpio_dev_data_t *pdata;
    /* ドライバデータ取り出し */
    pdata = (cqpio_dev_data_t*)pci_get_drvdata(dev);
    /* cqpio ドライバ削除 */
    unregister_cqpio(pdata);
    pci_release_regions(dev);
}
/*================================================*/
/* ドライバが対応する PCIデバイスの定義           */
/*================================================*/
/* ドライバが対応する PCIデバイスの定義 */                                       ┐
static struct pci_device_id cqpio_pci_tbl[] __devinitdata = {                     ├─Ⓑ
    { 0x6809, 0x8200, PCI_ANY_ID, PCI_ANY_ID, 0, 0, 0},                           │
    { 0 }                                                                         │
};                                                                                ┘
MODULE_DEVICE_TABLE(pci,cqpio_pci_tbl);
/* PCI バス・ドライバへの登録定義 */
static struct pci_driver cqpio_driver = {
    name:           "cqpio",
    id_table:       cqpio_pci_tbl,
    probe:          cqpio_probe,
    remove:         __devexit_p(cqpio_remove),
};
/*  */
static int __init init_cqpio_pci(void)            ┐
{                                                 │
    /* PCI ドライバへの登録 */                    ├─Ⓐ
    return pci_module_init(&cqpio_driver);        │
}                                                 ┘
/*  */
static void __exit exit_pci_cleanup(void)
{
    /* PCI ドライバからの登録解除 */
    pci_unregister_driver(&cqpio_driver);
}
/*  */
module_init(init_cqpio_pci);
module_exit(exit_pci_cleanup);
```

リスト5　コンパイルを実施するためのMakefile

```
CFLAGS= -D__KERNEL__ -I/usr/src/linux-2.4/include -I../include ¥
        -Wstrict-prototypes -O2 -fomit-frame-pointer ¥
        -fno-strict-aliasing -pipe -fno-strength-reduce -mcpu=i686 -DCPU=686 ¥
        -DMODULE -DMODVERSIONS ¥
        -include /usr/src/linux-2.4/include/linux/modversions.h¥
        -DEXPORT_SYMTAB

TARGET=cqpio.o cqpio_pci.o cqpio_cs.o

all: $(TARGET)

install:
    cp $(TARGET)  /lib/modules/`uname -r`/kernel/drivers/cqpio

clean:
    rm -f *.o
```

ルに定義が記述されています．cqpio_pci.o，cqpio_cs.oモジュールは，cqpio.oモジュール内の関数を呼び出します．このため，cqpio.oは，二つのモジュールよりも先にロードされている必要があります．modules.depファイルの記述によって依存関係を解決してドライバをロードできます．

PCIデバイス情報は，PCIデバイスのベンダIDやデバイスIDなどの情報とドライバ・モジュールの対応付けに必要です．modules.pcimapファイルに記述されています．

（4）PCMCIAドライバ登録

PCMCIAドライバは，CardBus（PCI）のようにdepmodコマンドでは登録できません．手動で/etc/pcmcia/cqpio.confファイルを追加します（リスト6）．

deviceセクションには，ドライバ・モジュールを記述します．cardセクションはカードのCISタプルの製品情報とdeviceセクションに定義したドライバの関連付けを記述します．

（5）デバイス・ファイル作成

PIOドライバ用のデバイス・ファイルを作成します．メジャー番号は240，マイナ番号にはゼロを使用します．作成には以下のコマンドを実行します．

```
# mknod c /dev/cqpio0 240 0
```

● 動作確認

（1）動作確認プログラム仕様

動作確認プログラムは，引き数を解釈してPIOを操作する各ioctlを発行する簡単なものです．引き数の仕様は以下のとおりです．

▶ -D

-Dはディップスイッチ設定を読み出し，1/0の4桁でコンソールに表示します．

▶ -L1100

-LはLEDの点灯・消灯を制御します．-Lに続けて4桁の1/0の文字列を与ます．

▶ -I

-Iを指定すると，割り込みボタンの押下をカウントします．

（2）動作確認プログラムの実行

動作確認プログラムの実行のようすを図4に示します．ドライバの仕様は，PCMCIAとCardBusは同じなので，実行のようすはまったく同じです．

リスト6　PCMCIAドライバの登録（/etc/pcmcia/cqpio.conf）

```
device "cqpio_cs"
  class "cqpio" module "cqpio_cs"

card "PIO PCCard(PCMCIA)"
    version "Kurusugawa-ele.", "PIO PCCard"
    bind "cqpio_cs"
```

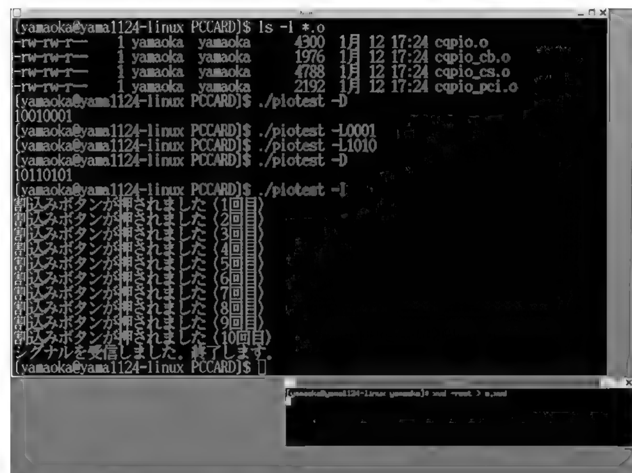


図4　実行結果

まとめ

今回作成したPCカード・ドライバはLinux 2.4向けに作成しましたが，すでに最新のカーネルはLinux 2.6に移行しています．しかし，ソース・コードを眺めた範囲では，PCカード関連に大幅な変更はないようです．本稿の内容はLinux 2.6にも流用できると思われます．

参考文献
（1）TECH I Vol.14 PCカード/メモリカードの徹底研究，CQ出版（株）．

やまおか・けんいち　（株）オーク

「VxWorks」を使ったRTOS技術の基礎と応用



第7回

続・RTOS再入門――デスクトップOSプログラムからの脱却

高山 剛

UNIXプログラマから組み込みシステム・エンジニアになるための基礎知識

● RTOS上でもANSI関数やUNIX互換APIを使ってプログラミング

前回，排他制御セマフォ，割り込み，同期，タイマ同期という RTOS特有の機能とAPIを紹介しました．いままでWindowsやUNIXなどのアプリケーションを書いていたエンジニアはまったく違う世界のように思ったかもしれませんが，実は排他制御セマフォ，割り込み，同期，タイマ同期などはアプリケーションの中のほんの一部に過ぎません．場合によってはシステム全体のコードの1%程度，もしくはずっと少ないかもしれません．

では残りの99%はどのようにコーディングすれば良いのでしょうか．もし，WindowsやUNIXと同様，よく知られたANSI関数，POSIX関数がRTOSにサポートされていれば，デスクトップ向けのプログラミングと同じようにコーディングができます．VxWorksの場合は，さらによく知られたUNIXライクなAPI(signal，socket，pipe，tty，ptty，select，ioctl，setenv/setjmp)，C++ I/O stream，STLなどをサポートしているため，RTOSを初めて使う場合でも，RTOS上でANSI関数やUNIX互換APIを使ってプログラミングを開始できます．

たとえば，

```
main()
{
  printf("Hello World¥n");
}
```

は，VxWorksでも動作します．

`printf()`の出力はUNIXと同様に標準出力に出力され，標準出力をリダイレクトすることでシリアル，VGAモニタ，さらにはTELNETのpttyにまで入出力できます．VxWorksはUNIXライクな軽量なI/Oシステムをもっているためです．

● 一定周期でのデータ・サンプリング

次に，少しRTOSらしいことをしてみましょう．

```
main()
{
unsigned char * io;
  io = PORT;
  for(;;)
   {
   data = *io;
   taskDelay( 60);
   printf("Data=%3d ¥n",data);
   }
  }
}
```

とすると1秒ごとに(デフォルトで60tick=1秒)，I/Oポートのデータを読み込み`printf`で数値を表示することができます．このように一定周期でのデータ・サンプリングが記述できました．

● 割り込み時にポートの内容を読んで表示

次は割り込みが起こったとき，即座にポートの内容を読んで表示するというコードを試します．

```
warikomi()
{
unsigned char * io;
  io = PORT;
  data = *io;
  printf("Data=%3d ¥n",data);
}
```

としたいところですが，割り込みサービス・ルーチンでは使えるAPIが決まっているという制限があって，OSのマニュアルに厳密に定義されています．基本的に，タスク・スケジューリングを誘発するタイプのAPIは使えません．

このため，VxWorksでは`printf()`の代わりに`logMsg()`というAPIを用意しています．`logMsg()`は`printf()`同様，フォーマット付きの標準出力を行う`printf`コンパチブルなAPIですが，`logMsg()`は渡された引き数を，何もせずメッセージ・キューで送信して特別なタスク(`excTask`)で受信して，こちらでフォーマット付きで標準出力します．

```
warikomi()
{
```

```
unsigned char * io;
  io = PORT;
  data = *io;
  logMsg("Data=%3d ¥n",data);
}
```

注意：初期化時に，

```
    intConnet( Vector, warikomi , 0 );
```

を呼び出し，割り込みベクタと割り込みサービス・ルーチンを接続しておくことが必要．

となります．このlogMsg()は割り込みのデバッグ時に強力で，次のようにあちらこちらにlogMsg()を埋めておいてデータの内容をモニタするときにたいへんちょうほうします．

```
warikomi()
{
int data;
  data = *PORT1;
  logMsg("P2=%3d ¥n",data);

  data = *PORT2;
  logMsg("P2=%3d ¥n",data);

  logMsg("P3=%3d ¥n",data);
  while( *P3 )
  {
  }
}
```

● タスクの生成方法

最後に,タスクの生成の方法を紹介します．タスクは関数taskSpawn()により生成します．

VxWorksは，ブート後にrootTaskと呼ばれるタスクが立ち上がってOSが各コンポーネント(Networkやファイル・システムなど)を初期化しますが，このタスクの最後にアプリケーションのタスクをtaskSpawnを通じて生成することでアプリケーション(タスク)を起動させることになります(図1)．

このようにUNIXを知っていれば，VxWorksのAPIと十数個を活用するだけで，とりあえず組み込み用アプリケーションを書くことができます．たとえ簡単なもので少し動いてしまえば，ほかの多くの便利なAPIを少しずつ習得することで本格的なアプリケーションを構築できるようになるでしょう．

イベント・ドリブン(駆動)プログラミング

今まで説明してきたRTOSの機能は，

▶ 割り込み→セマフォによる同期→タスク

▶タスク→セマフォによる同期→タスク(クライアント－サーバ・モデル)

▶ タイマ→タイマとの同期→タスク

などのように，イベント・ドリブンであることがわかると思います．ほかにもファイル・システムに対して行うread()/write()も，ソケットを介してのTCP/IPでのsend()/receive()/accept()もディスクやネットワークからの受信，送信可で割り込みが発生し，それに同期するのでイベント・ドリブンです．

実はOS内部ではセマフォやメッセージ・キューが使われ，ドライバやプロトコル・スタック，I/Oシステムがセマフォやメッセージ・キューでイベント・ドリブンで動作しているのです．

図2のように，RTOSを使った組み込みシステムは，イベント・ドリブンで動作するタスク群で構成されます．UNIXのエンジニアが組み込みシステムを設計する場合，イベント・ドリブンのプログラミングにまず慣れて，イベント・ドリブンでシステムを設計することが要求されます．頭の切り替えが少し必要です．

開発環境とは

さて，RTOSのAPIについては，全体像をつかんでいただけたと思います．しかし，これだけで組み込みシステムを作ることはできません．

UNIXであれば，まずエディタ，コンパイラ，リンカ，ウィンドウ・システムまで習得していれば十分なのですが，組み込みシステムの場合は多くの壁があります．

● 開発環境とターゲットでCPUが異なる

CPUリセット

romInit() リセット時のエントリ，CPU制御レジスタ，メモリコントローラ設定などCPU依存コードの作成

関数コール

romStart() Genericな初期化コード(data segment copy，bssクリア)の作成

関数コール

usrInit() 割り込みベクタ初期化，キャッシュ設定，ハードウェア初期化，カーネル初期化によりマルチタスクへ移行

タスク生成

usrRoot() カーネル初期化直後にタスクとして起動される．タイマ，I/Oシステムやファイル・システムなどのコンポーネントの初期化を行い，最後にアプリケーションのタスクを起動

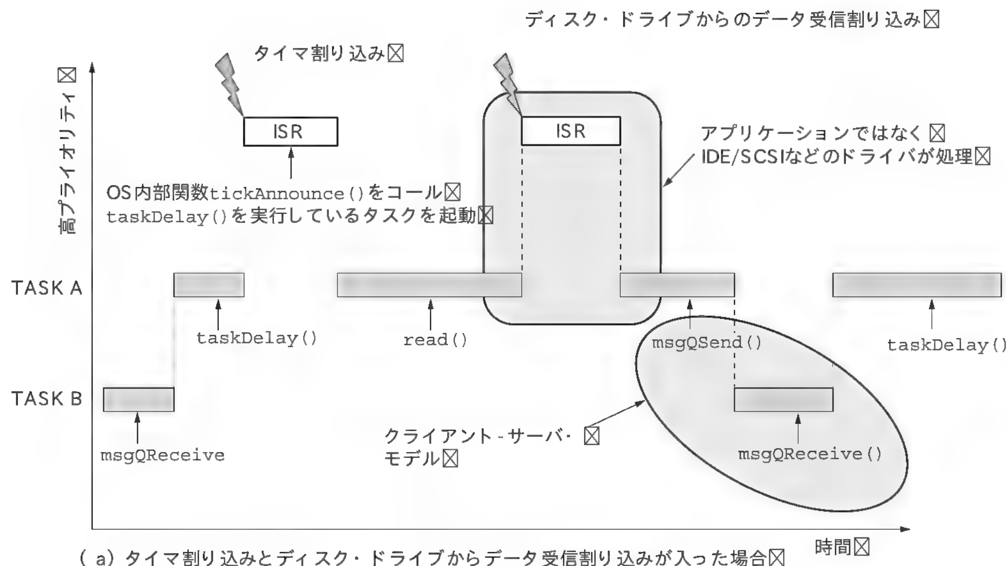
タスク生成

システム制御のメイン・タスク | タスク | タスク | 複数のタスクを起動可能

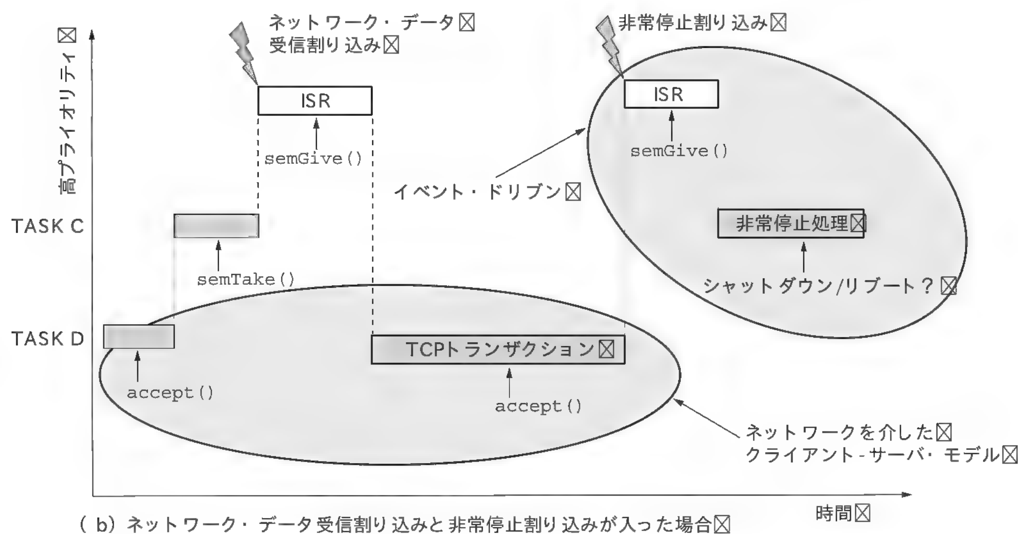
タスク生成

アプリケーションAのメイン・タスク | アプリケーションBのメイン・タスク

図1 CPUリセットからアプリケーション・タスクの起動まで



(a) タイマ割り込みとディスク・ドライブからデータ受信割り込みが入った場合



(b) ネットワーク・データ受信割り込みと非常停止割り込みが入った場合

図2　RTOSを使った組み込みシステムはイベント・ドリブンで動作するタスク群で構成される

- ターゲットのメモリが少ない
- 周辺機器が特殊
- ターゲットがディスク・ドライブをもっていない
- グラフィックス・ハードウェアをもっていない
- 自社開発ボードに移植しなければならない

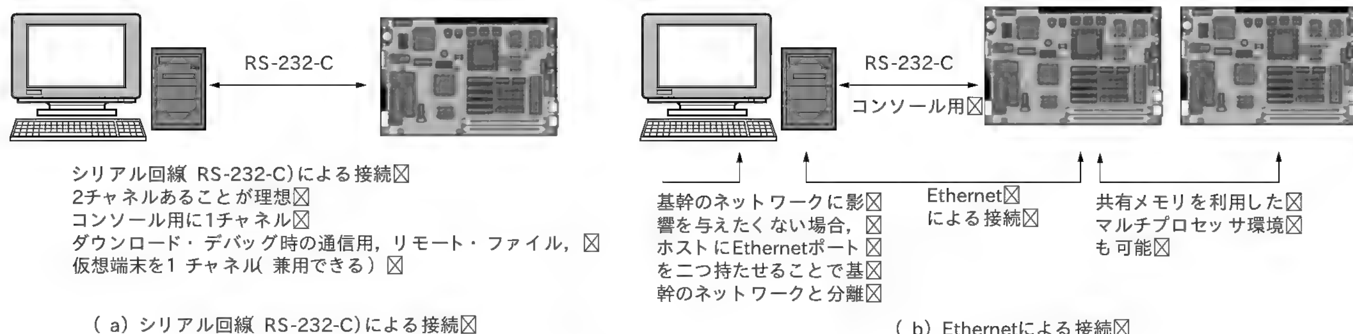
これらの理由により，組み込みソフトウェアの開発をデスクトップOSで行い，ターゲット・システムにプログラムをEthernetやICEを通じてダウンロードしてデバッグする環境を「クロス開発環境」と呼びます(図3).

クロス開発環境とは，クロス・コンパイル，リンク，ソース・コード・リビジョン管理，CM(コンフィグレーション・マネージメント，日本語では構成管理ツールと呼ばれる)，プロジェクト管理，静的コード解析，CASEツール，ソース・コード編集，VxWorksシミュレータなどをデスクトップ・パソコンで実行し，EthernetやICEを通じて通信することで，プログラムのダウンロード，修正，実行，シングル・ステップ，ソース・コード・レベル・デバッグ，システム状態の視覚化ができる環境をいいます.

VxWorksの場合，クロス開発環境がなくてターゲットしかない環境下でも強力なデバッグ・ツールであるTarget shellだけでのセルフ・デバッグも可能です.

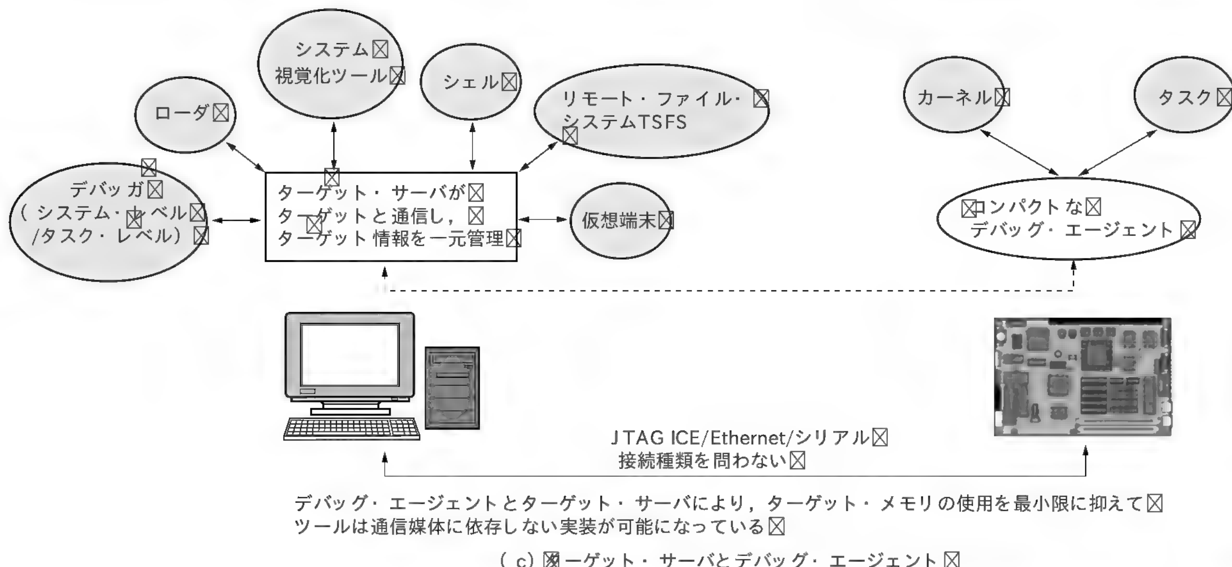
● ダウンロードはどうする？

クロス開発環境ではダウンロードの方法がもっとも問題です．かつては低速なシリアル・ケーブル(筆者は昔，たかだか200Kバイトのプログラムを5分もかけてダウンロードしていた)で接続してSレコード・フォーマットで転送する方法か，高価な

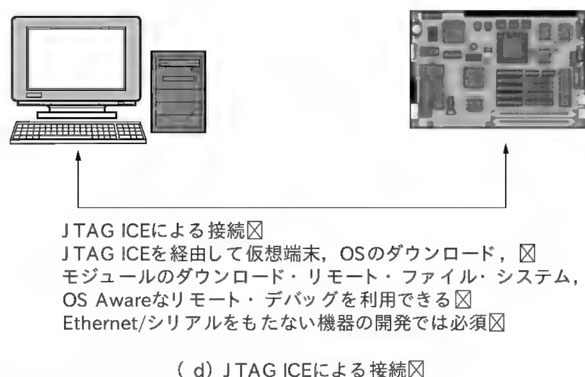


(a) シリアル回線(RS-232-C)による接続

(b) Ethernetによる接続



(c) ターゲット・サーバとデバッグ・エージェント



(d) JTAG ICEによる接続

図3 クロス開発環境のいろいろ

ICEを用いる例が多かったのですが，今は手軽で高速なEthernetが一般的です．Ethernetを使えば，ものの1秒でダウンロードできます．

● デスクトップOSのソフトウェア開発とは違うデバッグ手法

組み込みシステムでのデバッグは，デスクトップOSでのデバッグよりもたいへんです．組み込みシステムでは，ハードウェアも最先端の技術を使うのでハード的な問題，不完全なドライバの品質，さらに最先端のCPUを使うため，コンパイラが問題を抱えている場合もあります．

組み込みエンジニアは，不具合があった場合，これらが問題なのか，アプリケーションの問題なのかを切り分ける必要があります．そのためメモリのダンプ，逆アセンブル，タスクの状態表示，ヒープ・メモリの空き容量/最大ブロック・サイズの確認，ネットワーク・スタティックス，ping，バック・トレース(関数の呼び出しを過去にさかのぼる)，CPUレジスタの参照，修正，タスクのデリート/サスペンド(強制停止)，バッチ処理，実行時間の測定，シングル・ステップ/ステップ・オーバ，関数コール，グローバル変数参照，例外処理，割り込みのデバッグ，デバッグ・コードを後からダウンロードしてデバッグ，スタック最大消費量の確認，OSへのシステム・コールがエラーを起こした場合の根本原因を知る(error code)，周期的実行，繰り返し実行，Telnetからのリモート・ログイン，リモート・ファイル・システムへのダンプ，タイマの周期変更/停止，タスクごとのCPU使用率の測定，事後解析(Post mortem)，ワーム・リブート，`printf`，リダイレクトなどのあらゆる情報，ツールを組み合わせ，これらを駆使してのデ

Column 1

VxWorksのハードウェア要件

VxWorksを使う場合，どのくらいメモリを使うか，周辺機器に何が必要かと聞かれることがよくあるので簡単に紹介します．データは筆者が過去のバージョンのVxWorksで調べたものなので，目安と考えてください．

まず，産業機器など，メモリのコストやEthernetインターフェースのコストが問題にならない場合，OSにはカーネル・ネットワーク・プロトコル・スタック（効率的なデバッグやモニタリングを考慮して），telnetd，FTP，ファイル・システム（プログラム，画像，ティーチング・データの保存用），I/Oシステム，ドライバ（タイマ，Ethernet，シリアル，IDE）が含まれていればまず十分です．

この場合，RISCプロセッサでおおよそ700Kバイトのコード・サイズとなります．

ディジタル・コンシューマ機器でコード・サイズを最小に抑えたい場合，

- 最小構成のカーネル
- フラッシュ・ファイル・システム，I/Oシステム
- ドライバ（タイマ，フラッシュ）

のみであれば，大まかにコンフィグレーションして，おおよそ150Kバイト程度でした．さらに小さなコンポーネントを削除するなり，ソース・コードに手を入れるなどすれば，いくらでも小さくできるのですが，アプリケーションの使うAPIをどこまでと決められないのでこのあたりが妥当な線といえます．

ハードウェア要件としては，メモリが上記の程度を目安に最低限，タイマ1チャネルとEPROMまたはフラッシュ・メモリをハードウェアとして搭載すればよく，デバッグ用にシリアル，Ethernetがあれば便利といえます．CPUはMMUは不要で，FPUがなければソフトウェア浮動小数点演算ライブラリが提供されています．CPU内部が32ビットであれば，外部バスが何ビットかは問いません．

バッグが必要です．

VxWorksではターゲット・シェル，Tornado統合環境ではホスト・シェルによるコマンド・ラインによるデバッグからソース・コード・レベル・デバッグ，システム・レベル・デバッグによるドライバのデバッグ，ブラウジング機能，システム（トレース，カバレッジ，タスク，データ・フロー，ヒープ，CPU負荷）の視覚化を行うビジュアライゼーション・ツールやICEまでをサポートしています．

これらデバッグ手法については別の機会に詳しく解説していきたいと思います．

● テスト・チューニング

製品の差別化を図るのは機能，性能，コスト，信頼性ですが，カバレッジ・ツール，関数単位でのCPU負荷計測，関数呼び出し回数を計測するツールが信頼性，性能向上に必要となります．

OSがスケーラブル（必要な機能だけがきめ細かく組み込まれ，不要なモジュールを削除できる）であればサイズが抑えられ，コストの削減につながります．VxWorksはC言語のライブラリで提供され，OSやミドルウェアのコンポーネントをさらに細かくモジュール化しているため，不要なコードを取り払うのが簡単であるといった特徴をもっています．

● ROM化手法の手順

アプリケーションが完成すると，いよいよホスト・コンピュータなしにブートし，オペレーションなしでアプリケーションが動かなくてはなりません．一般には次のようなパターンがあります．

① ROM常駐

OSとアプリケーションがリンクされROMに書き込まれます．リセット後，まずプログラムのデータ・セグメント（初期値付きグローバル変数）がROMからRAMにコピーされ，BSSセグメント（初期値なしグローバル変数）をゼロ・クリアしてROM上のプログラムがそのまま動作します．この場合，起動は速くなりますが，ROMへのアクセスが遅い場合，アプリケーションとOSの実行が遅くなるという欠点があります．クイック・ブートが求められるディジタル・コンシューマの分野に向いています．

② ROM常駐RAM展開型

①と違い，プログラム自体もRAMにコピーされます．周波数の高いプロセッサを使用するアプリケーション（ネットワーク機器など）に向いています．

③ ROM常駐RAM展開型で圧縮

②の場合でROM使用量を節約したい場合に使用されます．

④ BootROM起動→OSをファイル・システム・ネットワークからロード→アプリケーション・コンポーネント単位でロード

フラッシュ・メモリの大容量低価格化で，フラッシュ・メモリ上にファイル・システムを搭載してOSアプリケーション・コンポーネント群を順次ロードしていきます．メンテナンス性が高く，今後多くのシステムで採用されるでしょう．

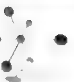
VxWorksでは，これらの構成がGUIやMake時にターゲットを選択することで可能になっています．

Pentium用のVxWorksではBIOSを使うので，BIOSによってOSをファイル・システムからロードするか，BIOS→BootROM→OSとロードすることもできます．

● マニュファクチャリングを考えて設計する

OSの話からはそれてしまいますが，ICEを導入しておくとハードウェアの立ち上げ，ソフトウェアのデバッグ，さらには製品のアセンブリ段階でも役立ちます．

たとえば，ROM化時のフラッシュ・メモリへのプログラム



の書き込み，ICEによるハードウェアのテストもスクリプト化による自動化でハードウェアの不良を検出できます．ソフトウェアでハードウェア診断テストも行うべきですが，その前に実行しておけば効率よく開発できます．

筆者は以前，ハードウェア担当エンジニアがVMEのバスを誤って一つずらして結線(VMEバスの不使用の信号線を専用の高速バスとしていたので)したがために不具合をソフトウェア上でデバッグして原因を追い，ようやくVMEバスの不具合にたどりついた経験があるので，あのときにこういう思想があればと記憶に残っていたのでとり上げました．

● 自社開発ボードへの移植

組み込みシステムの場合は，何でもありの世界です．他社製品に勝つためにはソフトウェアの一部をハードウェアで実現したりと，あらゆる試みが行われます．このような自社開発ボードでは，RTOSの移植性が重要です．VxWorksでは，OSとハードウェアのインターフェースのために，ボードごとに次のようなC言語のライブラリを準備しています．

このようなCコードや関連マニュアルをパッケージしたものをBSP(Board Support Package)と呼んでいます．

BSPを構成するC関数の例を次に紹介します．

- `sysEnetAddrSet()`――MACアドレスのセット
- `sysEnetAddrGet()`――MACアドレスの取得
- `sysMotFccEndLoad()`―Ethernet Driverのアタッチ
- `sysFccEnetEnable()`―Ethernet Driverを有効化
- `sysFccEnetDisable()`―Ethernet Driverを無効
- `sysHwInit()`――――ボード依存のハードウェア初期化
- `sysMemTop()`――――実装メモリの上限

などです．

BSPの移植は，初めてVxWorksを使う人がいきなり行う場合にはたいへん困難ですが，VxWorksでアプリケーションを組み，アプリケーションのデバッグを経験していれば，BSPの移植時のデバッグに生きてきます．

同一のCPUアーキテクチャのリファレンス・ボードを購入し(ほかのボード用のいろいろなBSPをオンライン・サポートからダウンロードできるので，BSPの書きかたについて参考になるだろう)，どのようにBSPが動いているか，初期化しているかがわかればアプリケーションをデバッグする要領でデバッグできるので，比較的簡単です．

というのは，シリアル(RS-232-C)さえ動けばTarget Shellを動かすことができるので，前述したデバッグ手法を駆使できるためです．そのためドライバの移植時に，割り込みサービスルーチン中に`logMsg`などを使ったり，I/Oのアドレスをマニュアルで直接リード/ライトしてデバッグできてしまいます．

Target Shellはたいへん強力です．たとえ不正アクセスでOSがあちこち壊れていても，カーネルのスケジューリングとシリアルのドライバが動いていれば，ある程度デバッグできてしまいます．

シリアルが動きTarget Shellが動けば，タイマのドライバを動かし，次にNetworksドライバを動かします．Networkドライバが動けば，BootROMを作成します．

実はBootROMは，VxWorksにダウンロード機能と非常に簡単なコマンド・ライン・インターフェースを付けただけのもので，実は「VxWorks＋アプリケーション」なのです．つまり，ネットワークが動けば，

Column2

アプリケーションからハードウェアをダイレクトにアクセスすることは良いのか悪いのか

デスクトップOSでは，アプリケーションからハードウェアを直接アクセスするのはご法度ですが，組み込みシステムやRTOSでは当たり前のことです．アプリケーションからハードウェアを直接アクセスできなくては，リアルタイム性能を発揮できない場合があり，大意でいえば，アプリケーション自体がドライバみたいなものという場合さえあります．

組み込みは何でもありの世界です．製品の性能が売れ行きを決めるのであれば，ドライバという抽象化もなし，極端にいえば，OSなしで，全部アセンブラにしても良いと思います．

しかし，アプリケーション→ドライバ→ハードウェアと抽象化すれば互換性を持ち，透過性のあるシステムが達成できます．将来新しいアーキテクチャのプロセッサやバスに乗り換えたり，新しい周辺機器やメディア(UPS10MからWireless，SCSIからUSB2)が登場すればドライバを置き換えたり，コンポーネントを置き換えるだけですぐに対応できるでしょう．

VxWorksはUNIXと違ってハードウェアをアクセスするのに必ずしもドライバやI/Oシステム，ファイル・システムを介してアクセスする必要はありません．アプリケーションから直接，I/Oをアクセスしてもかまいません．あくまで性能とコンパチビリティとトレードオフの関係にあり，アプリケーション依存なのです．

ブロック・デバイスや通信デバイスはドライバ化することで恩恵にあやかれますが，単なるA-D/D-A変換デバイス，DIOなどドライバにしても恩恵がない場合には，アプリケーションから直接アクセスすれば十分です．できれば独自のAPIでパフォーマンスやリアルタイム性を犠牲にしない程度に抽象化して透過性を実現する方向で努力すべきでしょう．

一度，ハードウェア・エンジニアにソフトウェアはどんな仕様変更があっても柔軟に対応できるように作っておくべきだといわれたことがありましたが，まったく予測してないことには対応できません．予測できない仕様変更までは対応できませんが，努力だけはしておきましょう．

Column3

VxWorks 6.0について

VxWorks 6.0(コードネームBase6)の事前評価プログラムが始まりました．

気軽に無料で参加できるので参加されてはいかがでしょう．VxWorksシミュレータとEclipseベースIDEでVxWorksと新しいメモリ・プロテクションRTP(Real Time Process)を試してはいかがでしょうか．

http://www.windriver.com/japan/announces/vxworks/release.html

```
Make bootrom
```

とコマンドを実行するだけでBootROMが作成でき，BootROMからNetwork経由でVxWorksをダウンロードできるようになります．

あとは各種ドライバを一つずつ移植していくことになりますが，BootROMが存在しているので，ICEなしでNetworkからブートできるようになり，より効率的に開発が進むでしょう．

最後に品質保証のため，Validation Test SuiteがTornadeに付属するので，各ドライバに対して，負荷テストや必須のI/Oコントロール・ファンクションをテストできます．ぜひとも一度テストすべきでしょう．

新しいタイプの組み込みソフトウェア開発手法

Javaの登場により，Write Once Run Everywhereというコンセプトで，デスクトップで開発済みのアプリケーションを組み込み機器でもハードウェアの違いを越えて動かせるようになりました．産業機器でもタイミングにクリティカルな処理は，C言語を使い，それ以外はJavaを使う場合があるようです．言語レベルでメモリの不正アクセスを防げるため，ソフトウェアの生産性が高まります．

HTMLというコンテンツに対してのブラウザ，ブラウザのプラグインとして有名なFLASHなど，組み込み機器にエンジンだけを搭載して，コンテンツをネットから取り込んで動かすことが当たり前になりました．

コンテンツだけを開発して組み込み機器自体は開発も変更も行わないシステムは，クロス開発とは呼べない新しい組み込み機器開発といえます．

おわりに

RTOSでのプログラミングだけでなく，自社開発ボードへの移植からROM化，開発環境，ハードウェア要件と組み込み開発の全体像を具体的に紹介することで，まず詳細に入り込むより全体像を把握していただけたかと思います．次回以降は，RTOSでのC言語，デバッグ手法やBSPの構築などについて，より詳細について紹介し，実際にVxWorksをすでに使っているユーザにも参考になるような連載にしたいと考えています．

たかやま・たけし　ウィンドリバー(株)
takeshi.takayama@windriver.com

やり直しのための信号数学 第25回

総まとめII（DFT，FFT編）

三谷 政昭



前回は，“正規直交基底ベクトルによる信号表現”を用いて，統一的な取り扱いによりDFT，DCT，パーシバルの定理などの信号処理全般に通じる普遍的な内容を説明した（少々，難解に感じられたかもしれない）．通り一遍ではなく，統一的な考えかたを知っておくことで，新たに出くわす直交変換にも柔軟に対応できるものである．

今回は，前回の内容をふまえ，「DFT，FFT編」と題して，まずDFT値（スペクトル）の物理的な意味，DFTの高速計算バージョン（FFT）を実現するうえでの土台となる考えかたを中心に説明する．なお，FFTの説明に際しては，従来の参考書でよく見受けられる行列による取り扱いではない方法での解説を試みた．（筆者）

DFTでわかる信号波形の性質

いま，大きさ6の直流成分と，周波数が10［Hz］で最大振幅が8，位相が$\pi/3$遅れたcos成分からなるアナログ信号$x(t)$，すなわち，

$$x(t)=6+8\cos\left(20\pi t-\frac{\pi}{3}\right) \quad \cdots\cdots(1)$$

を考えてみよう（図25.1）．最初に，アナログ信号を1/30［秒］ごとにサンプリングすると，ディジタル信号$\{x_k\}_{k=0}^{k=2}$として，

$$\begin{cases} x_0=x(0)=6+8\cos\left(-\dfrac{\pi}{3}\right)=10 \\ x_1=x\left(\dfrac{1}{30}\right)=6+8\cos\left(\dfrac{\pi}{3}\right)=10 \\ x_2=x\left(\dfrac{2}{30}\right)=6+8\cos(\pi)=-2 \end{cases} \quad \cdots\cdots(2)$$

が得られる．

ところで，$N=3$サンプルのディジタル信号に対するDFT値$\{G_\ell\}_{\ell=0}^{\ell=2}$は，

$$\begin{cases} G_0=\dfrac{1}{3}\left[x_0+x_1+x_2\right] \quad \cdots\cdots(3) \\ G_1=\dfrac{1}{3}\left[x_0+x_1W_3+x_2W_3^2\right] \quad \cdots\cdots(4) \\ G_2=\dfrac{1}{3}\left[x_0+x_1W_3^2+x_2W_3^4\right] \quad \cdots\cdots(5) \end{cases}$$

ただし，$W_3=e^{-j\frac{2\pi}{3}}=\cos\left(\dfrac{2\pi}{3}\right)-j\sin\left(\dfrac{2\pi}{3}\right)$

$$=-\frac{1}{2}-j\frac{\sqrt{3}}{2} \quad \cdots\cdots(6)$$

と定義されるので，式（2）の信号値を代入することにより，

$$\begin{cases} G_0=\dfrac{1}{3}[10+10+(-2)]=6 \quad \cdots\cdots(7) \\ G_1=\dfrac{1}{3}\left[10+10\times\left(-\dfrac{1}{2}-j\dfrac{\sqrt{3}}{2}\right)+(-2)\times\left(-\dfrac{1}{2}-j\dfrac{\sqrt{3}}{2}\right)^2\right] \\ \quad =2-j2\sqrt{3} \quad \cdots\cdots(8) \\ G_2=\dfrac{1}{3}\left[10+10\times\left(-\dfrac{1}{2}+j\dfrac{\sqrt{3}}{2}\right)^2+(-2)\times\left(-\dfrac{1}{2}-j\dfrac{\sqrt{3}}{2}\right)^4\right] \\ \quad =2+j2\sqrt{3} \quad \cdots\cdots(9) \end{cases}$$

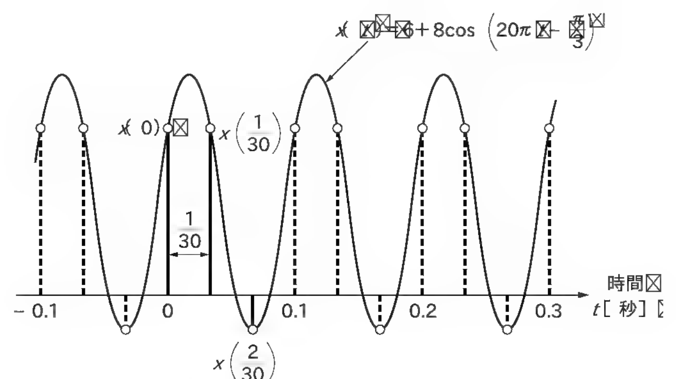


図25.1 アナログ信号とディジタル信号

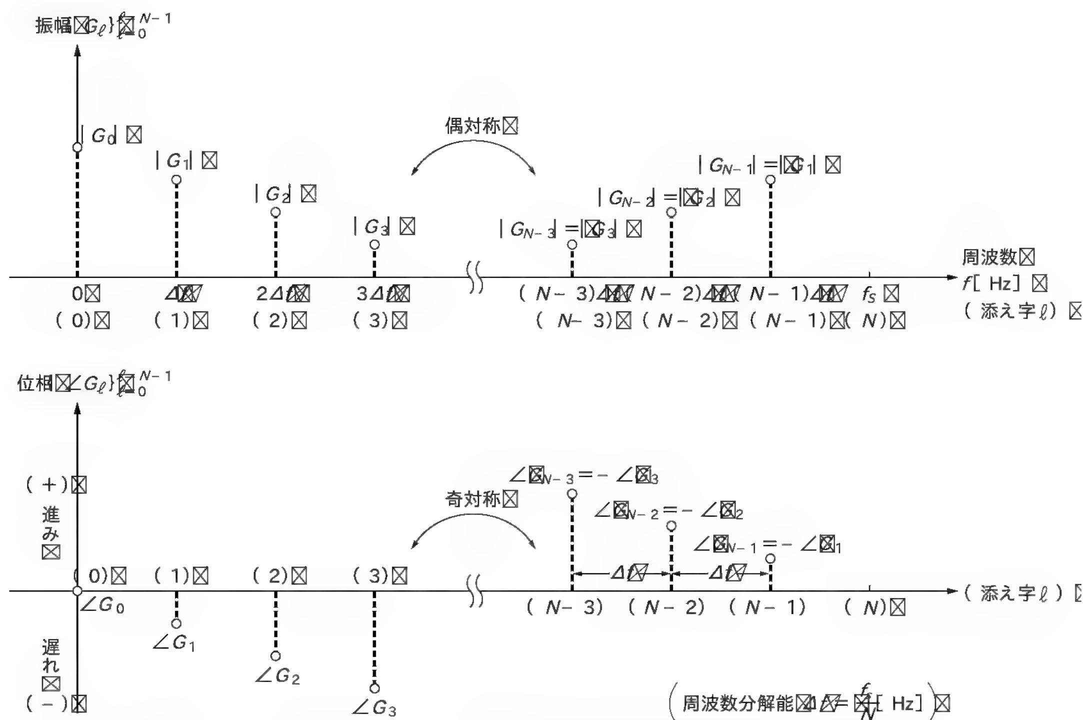


図25.2　実数値のディジタル信号に対応するDFT値$\{G_\ell\}_{\ell=0}^{\ell=N-1}$の特徴

と計算される．

また，サンプリング間隔Tは1/30[秒]なので，サンプリング周波数f_sは，

$$f_s = \frac{1}{T} = \frac{1}{\frac{1}{30}} = 30[\text{Hz}] \quad \cdots\cdots (10)$$

となる．よって，周波数分解能Δfは，

$$\Delta f = \frac{\text{サンプリング周波数}}{N} = \frac{30}{3} = 10[\text{Hz}] \quad \cdots\cdots (11)$$

である．

それでは，式(7)～式(11)から信号波形の性質をあぶり出してみよう．まず，式(7)の$G_0=6$と式(1)とを見比べることにより，直流成分を表していることに気づくし，G_0の添え字の"0"が0[Hz]という意味を表すと考えれば，すっきりと納得できる．

次は$G_1 = 2 - j2\sqrt{3}$であるが，複素数で表されたDFT値は必ず極形式で表現しないと，信号の性質を直接読み取ることができないことに注意してもらいたい．試しに，直交形式による表現($2 - j2\sqrt{3}$)を極形式に変換すると，

$$G_1 = 4e^{-j\frac{\pi}{3}} \quad \cdots\cdots (12)$$

となる．直流の場合と同様に考えて，式(12)を式(1)と比較することにより，次のことがわかる．

『G_1の絶対値(=4)を2倍した値(=8)は，最大振幅値を表す』

『G_1の偏角($-\pi/3$)は，位相を表す』

『G_1の添え字の"1"は，周波数分解能Δfの何倍かを表す』

『G_2は，G_1の複素共役($=\overline{G_1}$)に等しい』

これまで説明してきたことに基づき，Nサンプルの実数値のディジタル信号$\{x_k\}_{k=0}^{k=N-1}$に対するDFT値$\{G_\ell\}_{\ell=0}^{\ell=N-1}$が表す信号がもつ性質について，一般的にまとめておく(図25.2)．

(i)　周波数分解能$\Delta f = \frac{f_s}{N} = \frac{1}{NT}$[Hz] ……(13)

(ii)　G_ℓは$\ell\Delta f$[Hz]の周波数成分 ……(14)

(iii)　$G_{N-\ell}$はG_ℓの複素共役($=\overline{G_\ell}$) ……(15)

(iv)　最大振幅値はG_ℓの絶対値($=|G_\ell|$)と$G_{N-\ell}$の絶対値($=|G_{N-\ell}|$)の和，あるいは性質(iii)よりG_ℓの絶対値($=|G_\ell|$)の2倍 ……(16)

(v)　位相はG_ℓの偏角($=\angle G_\ell$) ……(17)

- 正(+)の値はcos波形を左にずらしたもの(進み位相)
- 負(−)の値はcos波形を右にずらしたもの(遅れ位相)

(vi)　基準のcos波形とのずれ時間

$$t_d = \frac{\text{位相}}{\text{角周波数}} = \frac{\angle G_\ell}{2\pi\ell\Delta f}[\text{秒}] \quad \cdots\cdots (18)$$

- 正(+)の値は進み時間
- 負(−)の値は遅れ時間

DFTはスペクトル分解のフィルタ群

ここではDFTの別な見かたを示すことにし，まずは結論を述べることから始めよう．



『DFT（ディジタル・フーリエ変換）は，フィルタ群（複数のフィルタを寄せ集めたもので，フィルタ・バンクという）であり，スペクトル分解機能を有する（図25.3）』

いま，Nサンプルのディジタル信号$\{x_k\}_{k=0}^{k=N-1}$に対するDFT値$\{G_\ell\}_{\ell=0}^{\ell=N-1}$は，

$$G_\ell=\frac{1}{N}\sum_{k=0}^{N-1}x_kW_N^{k\ell} \quad \cdots\cdots(19)$$

ただし，$W_N=e^{-j\frac{2\pi}{N}}=\cos\left(\frac{2\pi}{N}\right)-j\sin\left(\frac{2\pi}{N}\right) \quad \cdots\cdots(20)$

で与えられる．このとき，

$$V_N=W_N^{-1} \quad \cdots\cdots(21)$$

と置き，図25.3の各フィルタの伝達関数$\{H_\ell(z)\}_{\ell=0}^{\ell=N-1}$を，

$$H_\ell(z)=\frac{1}{N}V_N{}^\ell\sum_{m=0}^{N-1}V_N{}^{\ell m}z^{-m} \quad \cdots\cdots(22)$$

とすれば，DFT値を算出するシステムが構成される（図25.4）．また，各フィルタの入力$\{x_k\}_{k=0}^{k=\infty}$がNサンプルごとの繰り返し信号とし，その出力を$\{y_k{}^{(\ell)}\}_{k=0}^{k=\infty}$と表せば，式（22）より，

$$y_k{}^{(\ell)}=\frac{1}{N}V_N{}^\ell\sum_{m=0}^{N-1}V_N{}^{\ell m}x_{k-m} \quad \cdots\cdots(23)$$

で表される関係が成り立つ．最終的な出力$\{\tilde{y}_k{}^{(\ell)}\}_{k=0}^{k=\infty}$は，式（23）の出力値に対して，

$$\tilde{y}_k{}^{(\ell)}=\begin{cases}y_k{}^{(\ell)} & ;\ \ell=0\\ W_N{}^{(k \bmod N)+1}y_k{}^{(\ell)} & ;\ \ell\neq 0\end{cases} \quad \cdots\cdots(24)$$

を計算することにより，DFT値が得られる．なお，($k \bmod N$)は整数kをNで割り算したときの余りを示す．

それでは，N=3サンプルのディジタル信号を例に，DFT値がフィルタ・バンクの出力として得られることを検証してみよう．式（19）より，ディジタル信号$\{x_k\}_{k=0}^{k=2}$のDFT値$\{G_\ell\}_{\ell=0}^{\ell=2}$は式（3）～式（6）で計算できる．そこで，式（20）と式（21）より，

$$V_3=W_3^{-1}=e^{j\frac{2\pi}{3}}=-\frac{1}{2}+j\frac{\sqrt{3}}{2} \quad \cdots\cdots(25)$$

と置いて，

$$\begin{cases}H_0(z)=\frac{1}{3}\left[1+z^{-1}+z^{-2}\right] & \cdots\cdots(26)\\ H_1(z)=\frac{1}{3}V_3\left[1+V_3z^{-1}+V_3^2z^{-2}\right] & \cdots\cdots(27)\\ H_2(z)=\frac{1}{3}V_3^2\left[1+V_3^2z^{-1}+V_3^4z^{-2}\right] & \cdots\cdots(28)\end{cases}$$

で表される伝達関数を有するフィルタ・バンクを考える（図25.5）．

いま，図25.5のフィルタ・バンクの入力として図25.1に示すディジタル信号を考え，出力$\{y_k\}_{k=0}^{k=\infty}$を求めてみる．各フィルタの出力信号$\{y_k{}^{(\ell)}\}_{k=0}^{k=\infty}$は，図25.6～図25.8に基づいて次のように計算される．

① フィルタ$H_0(z)$の出力$\{y_k{}^{(0)}\}_{k=0}^{k=\infty}$（図25.6）

$$y_0{}^{(0)}=\frac{10}{3},\ y_1{}^{(0)}=\frac{20}{3},\ y_2{}^{(0)}=6,\ y_3{}^{(0)}=6,\cdots$$

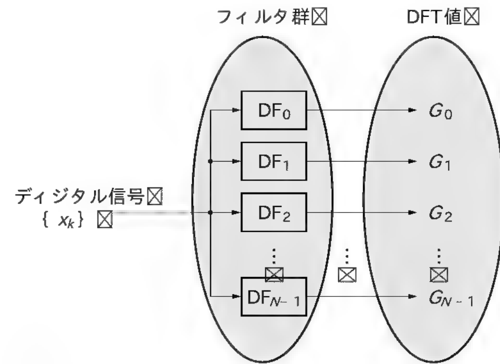


図25.3　DFTとフィルタ群

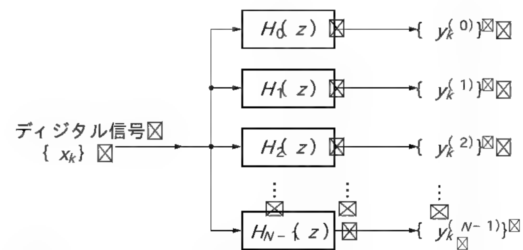


図25.4　フィルタ・バンク構成

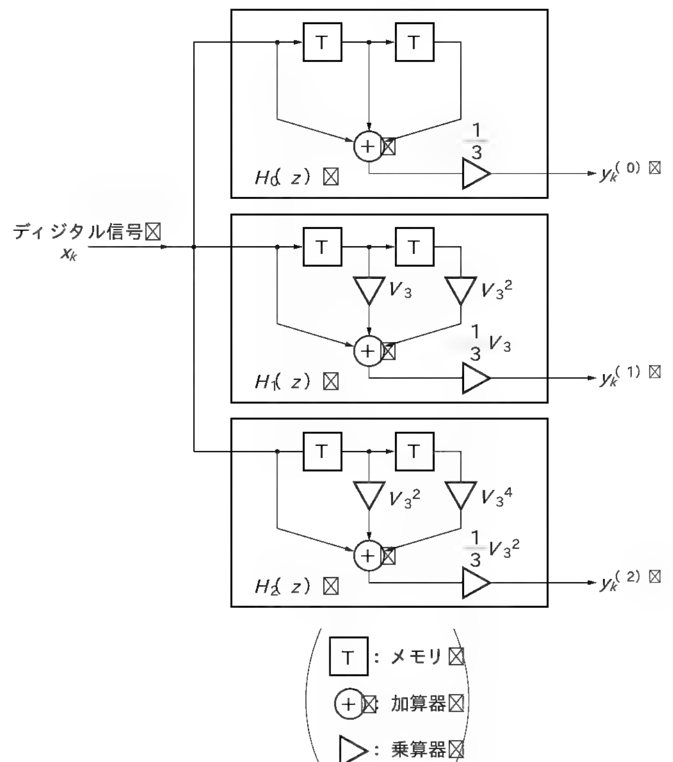


図25.5　DFTのフィルタ・バンク構成（N=3の場合）

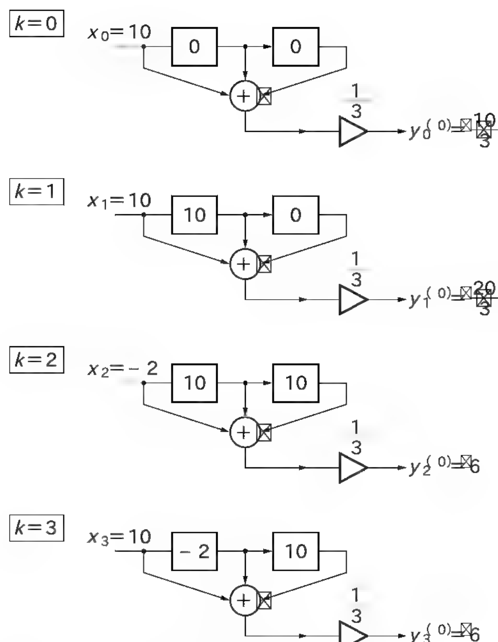


図 25.6 フィルタ $H_0(z)$ の出力計算 $\{y_k^{(0)}\}$

② フィルタ $H_1(z)$ の出力 $\{y_k^{(1)}\}_{k=0}^{k=\infty}$（図 25.7）

$$
\begin{cases}
y_0^{(1)} = -\dfrac{5}{3} + j\dfrac{5\sqrt{3}}{3} \\
y_1^{(1)} = -\dfrac{10}{3} \\
y_2^{(1)} = 2 - j2\sqrt{3} = 4e^{-j\frac{\pi}{3}} \\
y_3^{(1)} = 2 + j2\sqrt{3} = 4e^{j\frac{\pi}{3}} \\
y_4^{(1)} = -4 = 4e^{j\pi} \\
\vdots
\end{cases}
$$

③ フィルタ $H_2(z)$ の出力 $\{y_k^{(2)}\}_{k=0}^{k=\infty}$（図 25.8）

$$
\begin{cases}
y_0^{(2)} = -\dfrac{5}{3} - j\dfrac{5\sqrt{3}}{3} = \overline{y_0^{(1)}} \\
y_1^{(2)} = -\dfrac{10}{3} = \overline{y_1^{(1)}} \\
y_2^{(2)} = 2 + j2\sqrt{3} = 4e^{j\frac{\pi}{3}} = \overline{y_2^{(1)}} \\
y_3^{(2)} = 2 - j2\sqrt{3} = 4e^{-j\frac{\pi}{3}} = \overline{y_3^{(1)}} \\
y_4^{(2)} = -4 = 4e^{j\pi} = \overline{y_4^{(1)}} \\
\vdots
\end{cases}
$$

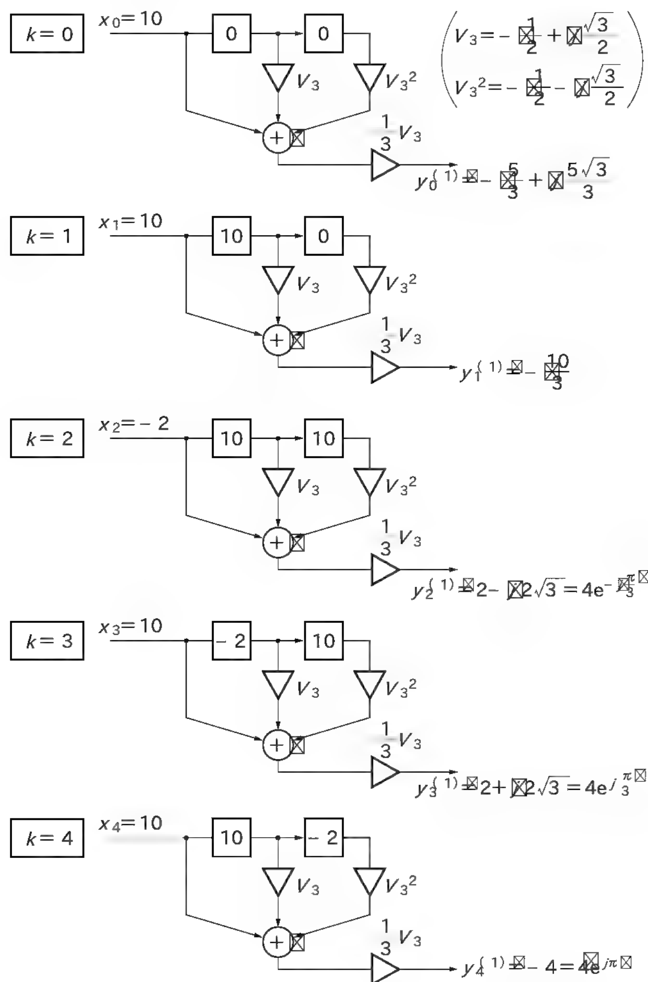


図 25.7 フィルタ $H_1(z)$ の出力計算 $\{y_k^{(1)}\}$

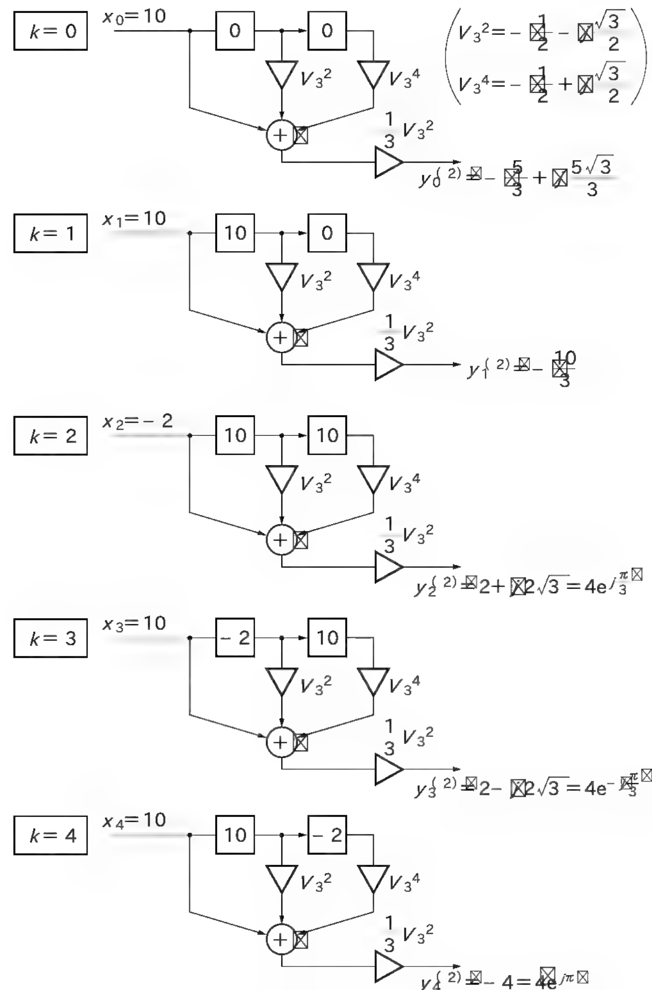


図 25.8 フィルタ $H_2(z)$ の出力計算 $\{y_k^{(2)}\}$



以上の結果において，各フィルタ出力のうち2サンプルまでの計算値$\left\{y_k^{(\ell)}\right\}_{k=0}^{k=1}$は過渡的な応答なので，意味のない出力であることに注意してもらいたい．また，フィルタ$H(z)$と$H(z)$の出力は複素共役の関係にあることも覚えておいてほしい．

さらに計算を進めて，式(24)より3サンプル以降の最終的な出力値は次のように計算される．

① 出力$\left\{\tilde{y}_k^{(0)}\right\}_{k=2}^{k=\infty}$

$$\tilde{y}_k^{(0)} = y_k^{(0)} = 6 \quad\cdots\cdots(29)$$

② 出力$\left\{\tilde{y}_k^{(1)}\right\}_{k=2}^{k=\infty}$

$$\tilde{y}_2^{(1)} = W_3^{(2 \bmod 3)+1} y_2^{(1)} = \underbrace{W_3^3}_{1} y_2^{(1)} = y_2^{(1)}$$

$$= 2 - j2\sqrt{3} = 4e^{-j\frac{\pi}{3}}$$

$$\tilde{y}_3^{(1)} = W_3^{(3 \bmod 3)+1} y_3^{(1)} = W_3^1 y_3^{(1)}$$

$$= e^{-j\frac{2\pi}{3}} \times \left(4e^{j\frac{\pi}{3}}\right) = 4e^{-j\frac{\pi}{3}}$$

$$\tilde{y}_4^{(1)} = W_3^{(4 \bmod 3)+1} y_4^{(1)} = W_3^2 y_4^{(1)}$$

$$= e^{-j\frac{4\pi}{3}} \times \left(4e^{j\pi}\right) = 4e^{-j\frac{\pi}{3}}$$

$$\vdots$$

$$\tilde{y}_k^{(1)} = 4e^{-j\frac{\pi}{3}} \quad\cdots\cdots(30)$$

③ 出力$\left\{\tilde{y}_k^{(2)}\right\}_{k=2}^{k=\infty}$

$$\tilde{y}_k^{(2)} = \overline{\tilde{y}_k^{(1)}} = 4e^{j\frac{\pi}{3}} \quad (\tilde{y}_k^{(1)}\text{の複素共役}) \quad\cdots\cdots(31)$$

このようにして得られた式(29)～式(31)の出力値はそれぞれ式(7)～式(9)のDFT値$\left\{G_\ell\right\}_{\ell=0}^{\ell=2}$に一致しており，フィルタ・バンクとDFT計算とが同じ機能を有することを実感できるのである．

例題1

式(29)～式(31)の各フィルタの出力値を見て，図25.5のフィルタ・バンクで分析した信号の性質を述べよ．

解答1

式(29)より，直流成分の振幅は6であることがわかる．式(30)の絶対値と偏角はそれぞれ，

$$\left|\tilde{y}_k^{(1)}\right| = \left|4e^{-j\frac{\pi}{3}}\right| = |4| \times \underbrace{\left|e^{-j\frac{\pi}{3}}\right|}_{1} = 4 \quad\cdots\cdots(32)$$

$$\angle\tilde{y}_k^{(1)} = -\frac{\pi}{3} \quad\cdots\cdots(33)$$

であり，最大振幅は8(=4×2)，位相の符号が負(－)なので“遅れている(基準となるcos波形を右にずらす)”ことがわかる．このときの遅れ時間は，式(18)より，

$$|t_d| = \left|\frac{-\frac{\pi}{3}}{2\pi \times 1 \times 10}\right| = \frac{1}{60}[\text{秒}] \quad\cdots\cdots(34)$$

となる．よって，10[Hz]のcos波形を1/60[秒]だけ右にずらしたものであり，1周期が1/10[秒]なので遅れ時間は1/6周期分に相当することもわかる(図25.9)．

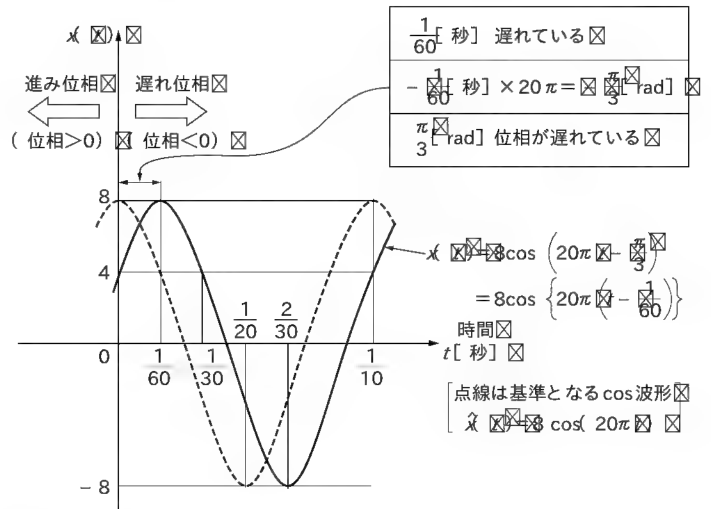


図25.9　DFT値に基づく波形情報の意味

DFTの高速算法の考え方

いま，$N=8(=2^3)$サンプルのディジタル信号$\left\{x_k\right\}_{k=0}^{k=7}$に対するDFTは，式(19)の定数部分(1/N)を除いた残りを$\left\{X_\ell\right\}_{\ell=0}^{\ell=7}$と表し，書き下すことで次のように表される．

$$\begin{cases} X_0 = x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\ X_1 = x_0 + x_1W_8^1 + x_2W_8^2 + x_3W_8^3 + x_4W_8^4 + x_5W_8^5 + x_6W_8^6 + x_7W_8^7 \\ X_2 = x_0 + x_1W_8^2 + x_2W_8^4 + x_3W_8^6 + x_4W_8^8 + x_5W_8^{10} + x_6W_8^{12} \\ \quad + x_7W_8^{14} \\ X_3 = x_0 + x_1W_8^3 + x_2W_8^6 + x_3W_8^9 + x_4W_8^{12} + x_5W_8^{15} + x_6W_8^{18} \\ \quad + x_7W_8^{21} \\ X_4 = x_0 + x_1W_8^4 + x_2W_8^8 + x_3W_8^{12} + x_4W_8^{16} + x_5W_8^{20} + x_6W_8^{24} \\ \quad + x_7W_8^{28} \\ X_5 = x_0 + x_1W_8^5 + x_2W_8^{10} + x_3W_8^{15} + x_4W_8^{20} + x_5W_8^{25} + x_6W_8^{30} \\ \quad + x_7W_8^{35} \\ X_6 = x_0 + x_1W_8^6 + x_2W_8^{12} + x_3W_8^{18} + x_4W_8^{24} + x_5W_8^{30} + x_6W_8^{36} \\ \quad + x_7W_8^{42} \\ X_7 = x_0 + x_1W_8^7 + x_2W_8^{14} + x_3W_8^{21} + x_4W_8^{28} + x_5W_8^{35} + x_6W_8^{42} \\ \quad + x_7W_8^{49} \end{cases} \quad\cdots\cdots(35)$$

ただし，$W_8 = e^{-j\frac{2\pi}{8}} = \cos\left(\frac{2\pi}{8}\right) - j\sin\left(\frac{2\pi}{8}\right)$

ここで，式(35)を直接計算したときの演算量は，

$$\begin{cases} \text{加算回数('+'記号の個数)} = 7 \times 8 = 56\text{回} \\ \text{乗算回数('}W_8^n\text{'の個数)} = 7 \times 7 = 49\text{回} \end{cases} \quad\cdots\cdots(36)$$

であるが，より少ない計算で済むように高速化を図ることを考えてみたい．演算量を削減するための基本的な考えかたは，式(35)において同一となる計算部分を見つけ出すことにある．

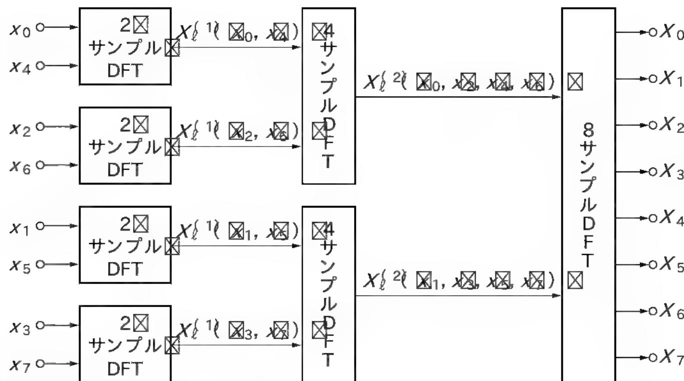


図25.10　DFTの多段構成(N=8の場合)

まず，式(35)の ℓ 番目のDFT値 G_ℓ(周波数 $\ell\Delta f$[Hz]のスペクトル成分に相当)は，

$$G_\ell = \frac{1}{N} X_\ell \quad \cdots\cdots(37)$$

$$\begin{aligned} X_\ell &= x_0 + x_1 W_8^{\ell} + x_2 W_8^{2\ell} + x_3 W_8^{3\ell} \\ &\quad + x_4 W_8^{4\ell} + x_5 W_8^{5\ell} + x_6 W_8^{6\ell} + x_7 W_8^{7\ell} \quad \cdots\cdots(38) \\ &= \left[x_0 + x_2 W_8^{2\ell} + x_4 W_8^{4\ell} + x_6 W_8^{6\ell}\right] \\ &\quad + W_8^{\ell}\left[x_1 + x_3 W_8^{2\ell} + x_5 W_8^{4\ell} + x_7 W_8^{6\ell}\right] \quad \cdots\cdots(39) \end{aligned}$$

と変形できる．さらに続けて，

$$\begin{aligned} X_\ell &= \left[x_0 + x_2 W_8^{2\ell} + x_4 W_8^{4\ell} + x_6 W_8^{2\ell} W_8^{4\ell}\right] \\ &\quad + W_8^{\ell}\left[x_1 + x_3 W_8^{2\ell} + x_5 W_8^{4\ell} + x_7 W_8^{2\ell} W_8^{4\ell}\right] \quad \cdots\cdots(40) \end{aligned}$$

となり，[　]の中を整理して，

$$\begin{aligned} X_\ell &= \left[\left(x_0 + x_4 W_8^{4\ell}\right) + W_8^{2\ell}\left(x_2 + x_6 W_8^{4\ell}\right)\right] \\ &\quad + W_8^{\ell}\left[\left(x_1 + x_5 W_8^{4\ell}\right) + W_8^{2\ell}\left(x_3 + x_7 W_8^{4\ell}\right)\right] \quad \cdots\cdots(41) \end{aligned}$$

と表される．

ところで，回転因子 W_8 には，

$$W_8^{\,2} = \left(e^{-j\frac{2\pi}{8}}\right)^2 = e^{-j\frac{2\pi}{4}} = W_4 \quad \cdots\cdots(42)$$

$$W_8^{\,4} = \left(e^{-j\frac{2\pi}{8}}\right)^4 = e^{-j\frac{2\pi}{2}} = W_2 \quad \cdots\cdots(43)$$

となる関係があるので，式(39)の[　]，式(41)の(　)の中はそれぞれ，

$$\begin{cases} X_\ell^{(2)}(x_0, x_2, x_4, x_6) = x_0 + x_2 W_4^{\ell} + x_4 W_4^{2\ell} + x_6 W_4^{3\ell} \\ X_\ell^{(2)}(x_1, x_3, x_5, x_7) = x_1 + x_3 W_4^{\ell} + x_5 W_4^{2\ell} + x_7 W_4^{3\ell} \end{cases} \quad \cdots\cdots(44)$$

$$\begin{cases} X_\ell^{(1)}(x_0, x_4) = x_0 + x_4 W_2^{\ell} \\ X_\ell^{(1)}(x_2, x_6) = x_2 + x_6 W_2^{\ell} \\ X_\ell^{(1)}(x_1, x_5) = x_1 + x_5 W_2^{\ell} \\ X_\ell^{(1)}(x_3, x_7) = x_3 + x_7 W_2^{\ell} \end{cases} \quad \cdots\cdots(45)$$

と表される．ここで，式(19)のDFTの計算式より，式(44)は4サンプルDFT，式(45)は2サンプルDFTであることが理解される．

したがって，式(39)，式(41)，式(44)，式(45)より，

$$\begin{cases} X_\ell^{(2)}(x_0, x_2, x_4, x_6) = X_\ell^{(1)}(x_0, x_4) + W_8^{2\ell} X_\ell^{(1)}(x_2, x_6) \\ X_\ell^{(2)}(x_1, x_3, x_5, x_7) = X_\ell^{(1)}(x_1, x_5) + W_8^{2\ell} X_\ell^{(1)}(x_3, x_7) \end{cases} \quad \cdots\cdots(46)$$

$$X_\ell = X_\ell^{(2)}(x_0, x_2, x_4, x_6) + W_8^{\ell} X_\ell^{(2)}(x_1, x_3, x_5, x_7) \quad \cdots\cdots(47)$$

と表され，DFTが“入れ子”形式の多段構成として得られる(図25.10)．

以上のように，DFTの高速化の真髄は，式(38)から式(41)の関係を導き出すプロセスにあるのだが，ピンとこないかもしれない．そこで，式(41)を書き下すことにより，DFTの高速計算アルゴリズムである“FFT”算法を体感してもらおうと思う．

つまり，式(41)は，

$$\begin{cases} X_0 = \left[\left(x_0 + x_4 W_8^{0}\right) + W_8^{0}\left(x_2 + x_6 W_8^{0}\right)\right] \\ \qquad + W_8^{0}\left[\left(x_1 + x_5 W_8^{0}\right) + W_8^{0}\left(x_3 + x_7 W_8^{0}\right)\right] \\ X_1 = \left[\left(x_0 + x_4 W_8^{4}\right) + W_8^{2}\left(x_2 + x_6 W_8^{4}\right)\right] \\ \qquad + W_8^{1}\left[\left(x_1 + x_5 W_8^{4}\right) + W_8^{2}\left(x_3 + x_7 W_8^{4}\right)\right] \\ X_2 = \left[\left(x_0 + x_4 W_8^{8}\right) + W_8^{4}\left(x_2 + x_6 W_8^{8}\right)\right] \\ \qquad + W_8^{2}\left[\left(x_1 + x_5 W_8^{8}\right) + W_8^{4}\left(x_3 + x_7 W_8^{8}\right)\right] \\ X_3 = \left[\left(x_0 + x_4 W_8^{12}\right) + W_8^{6}\left(x_2 + x_6 W_8^{12}\right)\right] \\ \qquad + W_8^{3}\left[\left(x_1 + x_5 W_8^{12}\right) + W_8^{6}\left(x_3 + x_7 W_8^{12}\right)\right] \\ X_4 = \left[\left(x_0 + x_4 W_8^{16}\right) + W_8^{8}\left(x_2 + x_6 W_8^{16}\right)\right] \\ \qquad + W_8^{4}\left[\left(x_1 + x_5 W_8^{16}\right) + W_8^{8}\left(x_3 + x_7 W_8^{16}\right)\right] \\ X_5 = \left[\left(x_0 + x_4 W_8^{20}\right) + W_8^{10}\left(x_2 + x_6 W_8^{20}\right)\right] \\ \qquad + W_8^{5}\left[\left(x_1 + x_5 W_8^{20}\right) + W_8^{10}\left(x_3 + x_7 W_8^{20}\right)\right] \\ X_6 = \left[\left(x_0 + x_4 W_8^{24}\right) + W_8^{12}\left(x_2 + x_6 W_8^{24}\right)\right] \\ \qquad + W_8^{6}\left[\left(x_1 + x_5 W_8^{24}\right) + W_8^{12}\left(x_3 + x_7 W_8^{24}\right)\right] \\ X_7 = \left[\left(x_0 + x_4 W_8^{28}\right) + W_8^{14}\left(x_2 + x_6 W_8^{28}\right)\right] \\ \qquad + W_8^{7}\left[\left(x_1 + x_5 W_8^{28}\right) + W_8^{14}\left(x_3 + x_7 W_8^{28}\right)\right] \end{cases} \quad \cdots\cdots(48)$$

と表され，式(36)より，

$$W_8^{\,4} = \left(e^{-j\frac{2\pi}{8}}\right)^4 = e^{-j\pi} = -1 \quad \cdots\cdots(49)$$

$$W_8^{\,8} = \left(e^{-j\frac{2\pi}{8}}\right)^8 = e^{-j2\pi} = 1 \quad \cdots\cdots(50)$$

となる．さらに，

$$\begin{cases} W_8^{\,5} = \underbrace{W_8^{\,4}}_{-1} \times W_8^{\,1} = -W_8^{\,1} \\ W_8^{\,6} = \underbrace{W_8^{\,4}}_{-1} \times W_8^{\,2} = -W_8^{\,2} \\ W_8^{\,7} = \underbrace{W_8^{\,4}}_{-1} \times W_8^{\,3} = -W_8^{\,3} \end{cases}$$

の関係も利用して式(48)を書き直すと，



$$
\begin{cases}
X_0=\left[(x_0+x_4)+(x_2+x_6)\right]+\left[(x_1+x_5)+(x_3+x_7)\right]\\
X_1=\left[(x_0-x_4)+W_8^{\,2}(x_2-x_6)\right]+W_8^{\,1}\left[(x_1-x_5)+W_8^{\,2}(x_3-x_7)\right]\\
X_2=\left[(x_0+x_4)-(x_2+x_6)\right]+W_8^{\,2}\left[(x_1+x_5)-(x_3+x_7)\right]\\
X_3=\left[(x_0-x_4)-W_8^{\,2}(x_2-x_6)\right]+W_8^{\,3}\left[(x_1-x_5)-W_8^{\,2}(x_3-x_7)\right]\\
X_4=\left[(x_0+x_4)+(x_2+x_6)\right]-\left[(x_1+x_5)+(x_3+x_7)\right]\\
X_5=\left[(x_0-x_4)+W_8^{\,2}(x_2-x_6)\right]-W_8^{\,1}\left[(x_1-x_5)+W_8^{\,2}(x_3-x_7)\right]\\
X_6=\left[(x_0+x_4)-(x_2+x_6)\right]-W_8^{\,2}\left[(x_1+x_5)-(x_3+x_7)\right]\\
X_7=\left[(x_0-x_4)-W_8^{\,2}(x_2-x_6)\right]-W_8^{\,3}\left[(x_1-x_5)-W_8^{\,2}(x_3-x_7)\right]
\end{cases}
\quad\cdots\cdots(51)
$$

となり，同じ計算部分を共通化することにより，多段構成の形式が見えてくる．

●第1段目

$$
\begin{cases}
x_0+x_4=A,\quad x_2+x_6=B,\\
x_1+x_5=C,\quad x_3+x_7=D,\\
x_0-x_4=E,\quad W_8^{\,2}(x_2-x_6)=F,\\
x_1-x_5=G,\quad W_8^{\,2}(x_3-x_7)=H
\end{cases}
\quad\cdots\cdots(52)
$$

●第2段目

$$
\begin{cases}
A+B=I,\quad C+D=J,\\
E+F=K,\quad W_8^{1}(G+H)=L,\\
A-B=M,\quad W_8^{2}(C-D)=N,\\
E-F=O,\quad W_8^{3}(G-H)=P
\end{cases}
\quad\cdots\cdots(53)
$$

●第3段目

$$
\begin{cases}
X_0=I+J,\quad X_1=K+L,\\
X_2=M+N,\ X_3=O+P,\\
X_4=I-J,\quad X_5=K-L,\\
X_6=M-N,\ X_7=O-P
\end{cases}
\quad\cdots\cdots(54)
$$

そこで，式(52)～式(54)に基づいて構成したDFT計算のブロック図を図25.11に示す．この多段構成による高速計算アルゴリズムにおいては，

加算回数(‘＋’，‘－’記号の個数)＝8×3＝24回

乗算回数(‘W_8^n’の個数)＝2+3＝5回

であることから，直接計算したときの演算量［式(36)］を大幅に削減できることがおわかりいただけるであろう．なお，これまで説明した“共通計算の省略”と“多段構成”という考え方を適用することにより，2のべき乗のサンプル数に対するFFT構成は，容易に一般化することができる．

一方，DFTの逆変換(IDFT)は，

$$
x_k=G_0+G_1W_8^{-k}+G_2W_8^{-2k}+G_3W_8^{-3k}+G_4W_8^{-4k}+G_5W_8^{-5k}+G_6W_8^{-6k}+G_7W_8^{-7k}\quad\cdots\cdots(55)
$$

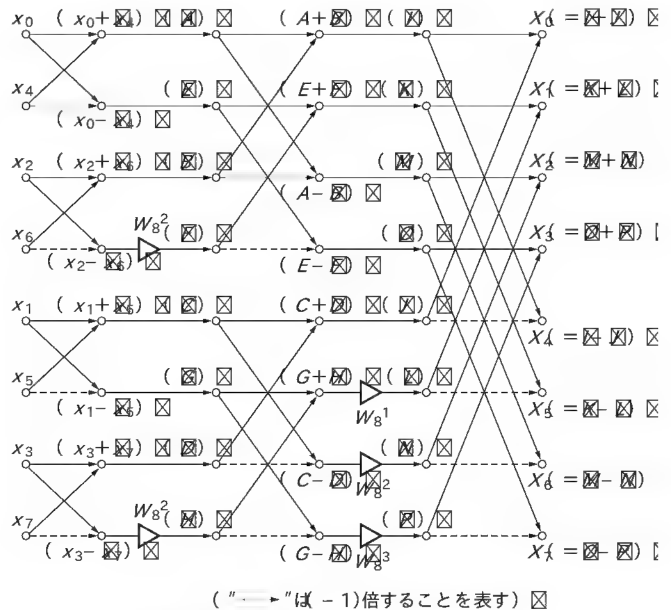


図25.11　FFT構成のブロック図

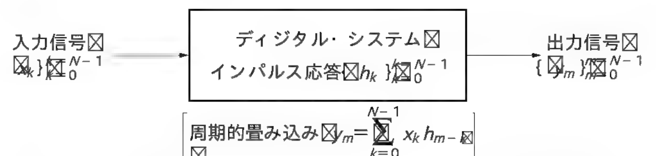


図25.12　ディジタル・システムの入出力関係(周期Nサンプル)

であり，式(37)，式(38)と見くらべれば，回転因子W_8^nをW_8^{-n}に置き換えるだけでDFT計算と同じになることが理解される．

周期的畳み込みとDFT

あるディジタル・システムのインパルス応答を$\{h_k\}_{k=0}^{k=N-1}$とするとき，入力信号$\{x_k\}_{k=0}^{k=N-1}$に対する応答出力$\{y_m\}_{m=0}^{m=N-1}$は，

$$
y_m=\sum_{k=0}^{N-1}x_{m-k}h_k\quad\cdots\cdots(56)
$$

あるいは，

$$
y_m=\sum_{k=0}^{N-1}x_k h_{m-k}\quad\cdots\cdots(57)
$$

と表される(図25.12)．

式(56)と式(57)は“時間領域での周期的畳み込み”とよばれ，x_{m-k}，h_k，x_k，h_{m-k}の添え字kと$(m-k)$はそれぞれ，

$$
\begin{cases}
k\quad\to k \bmod N\\
(m-k)\to(m-k)\bmod N
\end{cases}
\quad\cdots\cdots(58)
$$

のように，サンプル数N［個］で割り算した余りとして与える．この周期的畳み込み演算の意味について$N=4$サンプルを例に2通りで説明するが，どちらで理解してもかまわない．

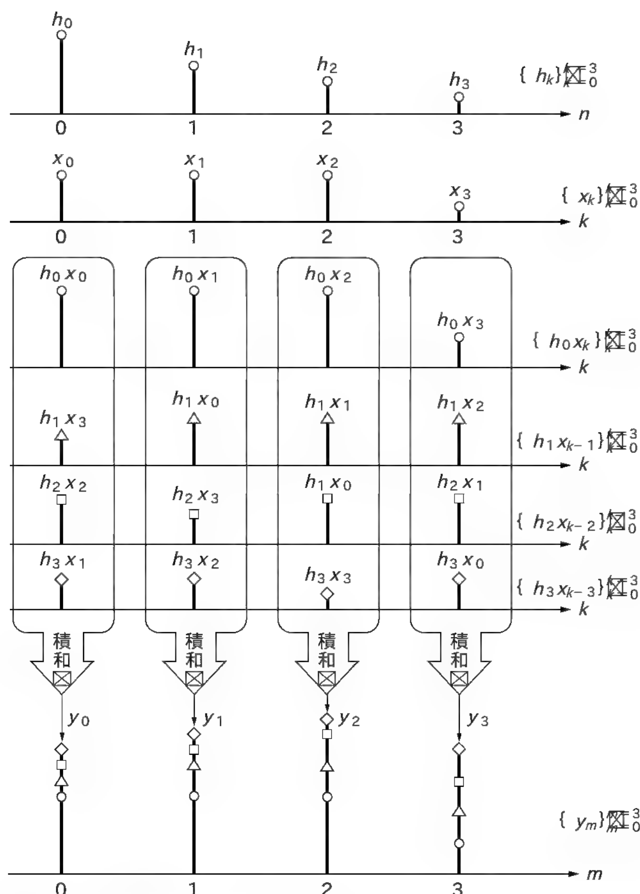


図 25.13　周期的畳み込みの説明(1)

● その1

式(56)に基づき，入力x_kを時間シフト(右に1サンプルずつずらすこと)しながら，インパルス応答h_kを掛けて総和を計算するもので，そのようす(積和という)を図25.13に図式的に示す．

● その2

式(57)に基づき，インパルス応答h_kの時間軸を反転して得られる$\{h_{-k}\}_{k=0}^{k=N-1}$を時間シフトしながら，入力x_kとの積和を計算する(図25.14)．

また，インパルス応答$\{h_n\}_{n=0}^{n=N-1}$，入力$\{x_k\}_{k=0}^{k=N-1}$，出力$\{y_m\}_{m=0}^{m=N-1}$のDFT値をそれぞれ，$\{H_\ell\}_{\ell=0}^{\ell=N-1}$，$\{X_\ell\}_{\ell=0}^{\ell=N-1}$，$\{Y_\ell\}_{\ell=0}^{\ell=N-1}$として，式(19)に基づき，式(56)あるいは式(57)の両辺をDFTすると，

$$Y_\ell = NX_\ell H_\ell \quad \cdots\cdots (59)$$

となる関係が成立する(詳細は，2002年2月号「DFTによるいろいろな信号分析」を参照)．

よって，周期的畳み込みとDFTとの関係，すなわち式(56)，式(57)と式(59)から，図25.12のディジタル・システムと同等

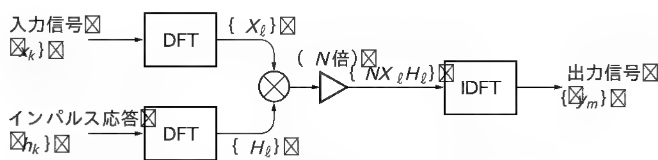


図 25.15　DFT，IDFTによるディジタル・システムの構成

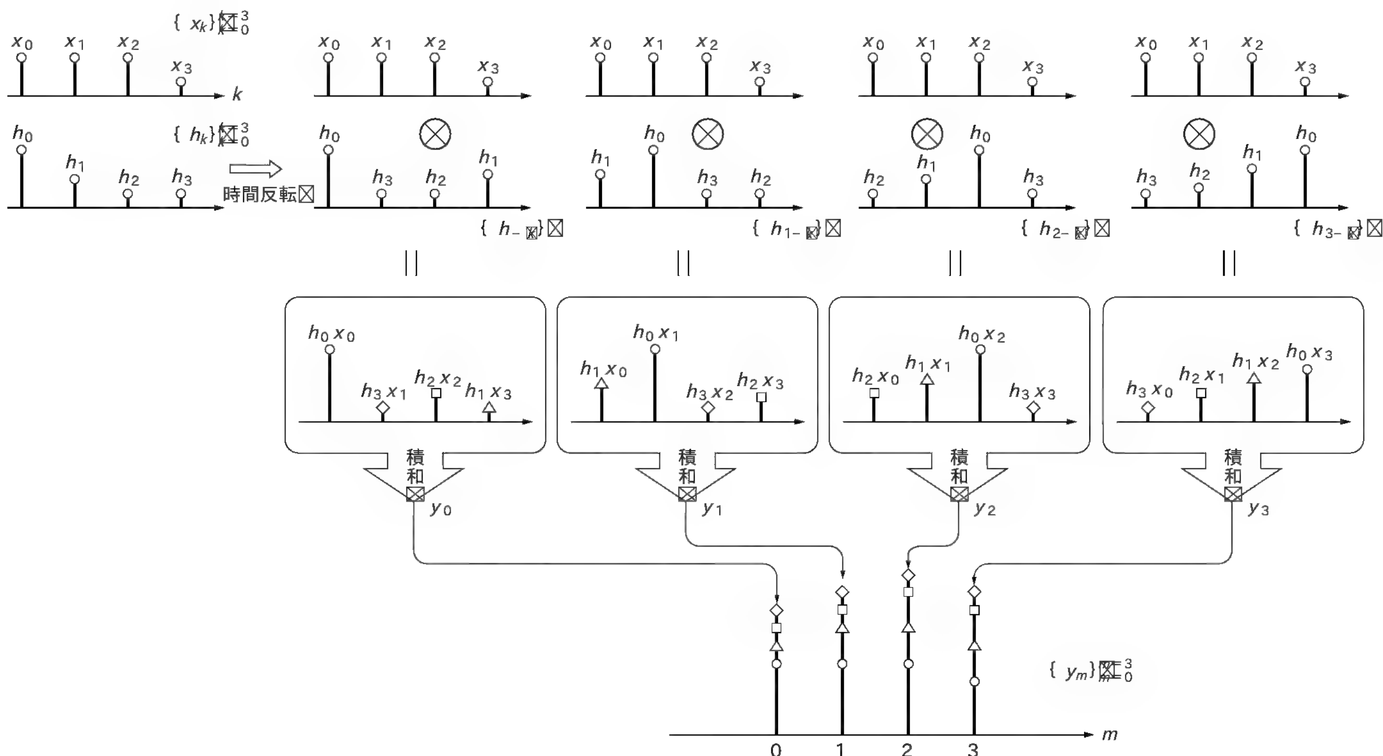


図 25.14　周期的畳み込みの説明(2)



の処理が，DFT と IDFT の組み合わせによって実行されることになる(図25.15)．

つまり，インパルス応答の DFT 値 $\{H_\ell\}_{\ell=0}^{\ell=N-1}$ と入力信号の DFT 値 $\{X_\ell\}_{\ell=0}^{\ell=N-1}$ の積の N 倍した値 $\{NX_\ell H_\ell\}_{\ell=0}^{\ell=N-1}$ を IDFT 計算することで出力応答が算出される．このとき，インパルス応答 $\{h_n\}_{n=0}^{n=N-1}$ の DFT 値 $\{H_\ell\}_{\ell=0}^{\ell=N-1}$ として，希望のスペクトル特性(たとえば，ローパス，ハイパスなどの周波数選択特性)を設定すれば，雑音除去や輪郭抽出などの信号処理を実現できる．

例題2

いま，図25.16(a)のインパルス応答を有するシステムに，同図(b)のディジタル信号を入力したときの出力信号 $\{y_m\}_{m=0}^{m=2}$ を求めよ．さらに，インパルス応答，入出力信号の DFT 値を計算し，式(59)の関係が成立することを示せ．

解答2

N=3として，式(57)を適用して出力を求める．以下に，式(58)の mod 演算に注意して計算した結果を示しておくので，検証しておいてほしい．

$$\begin{cases} y_0 = x_0h_0 + x_1h_{-1} + x_2h_{-2} = x_0h_0 + x_1h_2 + x_2h_1 = 22 \\ y_1 = x_0h_1 + x_1h_0 + x_2h_{-1} = x_0h_1 + x_1h_0 + x_2h_2 = 15 \\ y_2 = x_0h_2 + x_1h_1 + x_2h_0 = 11 \end{cases} \quad \cdots\cdots (60)$$

また，それぞれの信号の DFT 値は式(3)～式(6)に基づき，

$$\begin{cases} X_0 = \dfrac{1}{3}(-1+5+4) = \dfrac{8}{3} \\ X_1 = \dfrac{1}{3}\left[-1+5\left(-\dfrac{1}{2}-j\dfrac{\sqrt{3}}{2}\right)+4\left(-\dfrac{1}{2}+j\dfrac{\sqrt{3}}{2}\right)\right] \\ \quad = -\dfrac{11}{6}-j\dfrac{\sqrt{3}}{6} \\ X_2 = \overline{X_1} = -\dfrac{11}{6}+j\dfrac{\sqrt{3}}{6} \end{cases}$$

$$\begin{cases} H_0 = \dfrac{1}{3}(1+2+3) = 2 \\ H_1 = \dfrac{1}{3}\left[1+2\left(-\dfrac{1}{2}-j\dfrac{\sqrt{3}}{2}\right)+3\left(-\dfrac{1}{2}+j\dfrac{\sqrt{3}}{2}\right)\right] \\ \quad = -\dfrac{1}{2}+j\dfrac{\sqrt{3}}{6} \\ H_2 = \overline{H_1} = -\dfrac{1}{2}-j\dfrac{\sqrt{3}}{6} \end{cases}$$

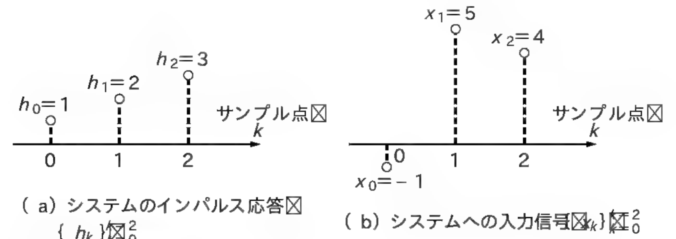


図25.16 例題2

$$\begin{cases} Y_0 = \dfrac{1}{3}[22+15+11] = 16 \\ Y_1 = \dfrac{1}{3}\left[22+15\times\left(-\dfrac{1}{2}-j\dfrac{\sqrt{3}}{2}\right)+11\times\left(-\dfrac{1}{2}+j\dfrac{\sqrt{3}}{2}\right)\right] \\ \quad = 3-j\dfrac{2\sqrt{3}}{3} \\ Y_2 = \overline{Y_1} = 3+j\dfrac{2\sqrt{3}}{3} \end{cases}$$

となり，

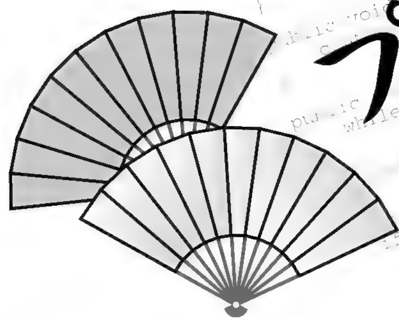
$$\begin{cases} 3X_0H_0 = 3\times\dfrac{8}{3}\times 2 = 16 = Y_0 \\ 3X_1H_1 = 3\times\left(-\dfrac{11}{6}-j\dfrac{\sqrt{3}}{6}\right)\times\left(-\dfrac{1}{2}+j\dfrac{\sqrt{3}}{6}\right) \\ \qquad = 3-j\dfrac{2\sqrt{3}}{3} = Y_1 \\ 3X_2H_2 = 3\overline{X_1}\,\overline{H_1} = 3\overline{X_1H_1} = 3\overline{Y_1} = Y_2 \end{cases}$$

であることから，式(59)が成立することを確認できる．

*　　　*

次回は，信号数学の総まとめの第3弾として，2次元データ(画像)を対象とした“DCT”を採り上げ，その物理的意味を中心にわかりやすく解説する予定である．お楽しみに．

みたに・まさあき　東京電機大学工学部情報通信工学科



プログラミングの要



宮坂 電人

第13回 バイナリ・ツリーとヒープ

サンプル・プログラムについて

前回まではサンプル・プログラムをJavaで記述していたのですが，今回からはC++で記述します．というのも型がからむ場合，残念なことにJavaではキャストにからむ問題が起こります（C言語はいわずもがな）．ところがC++ならばテンプレートで弊害を予防しやすくなります（Javaでも，間もなく登場するバージョンではジェネリックスというC++のテンプレートとほぼ同じしくみが利用できるようになる）．また動的配列の実装で`realloc`的なことができないため，今回紹介するヒープ構造のようなものが説明しにくいと判断したからです．それにからんで今まで記述してきたアルゴリズムのサンプル・プログラムをC++で書き直したものを用意したので，CD-ROMが付属する号（次号予定）を参照してください．

なお，記事で作成するC++のコードはMac OS X version 10.3.3のTerminalでgcc version 3.3を使って開発しましたが，標準ライブラリのみを使い，特定のOSに依存しないよう配慮しています．

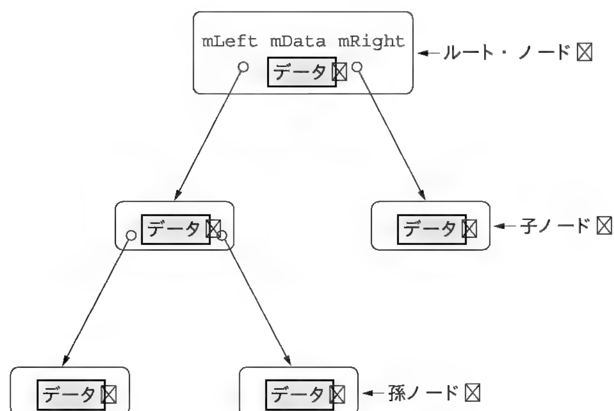


図1　バイナリ・ツリー

注1：`http://www.oreilly.com/catalog/masteralgoc/`を参照．
注2：ノードの左側にも右側にも別ノードが連結されていないノードのこと．

バイナリ・ツリー（Binary Tree）

ここまで紹介してきたデータ構造は複数のデータやオブジェクトを直線的に格納するもの（連結リストなど），あるいは無節操に格納するもの（ハッシュ・テーブルなど）でした．それらに対し，今回紹介するバイナリ・ツリー（「二分木」と呼ばれることもある）は，格納するデータに親子関係をつけ，左右方向に子供を連結していく構造です（図1）．バイナリ・ツリーを利用する場面にはいろいろ考えられますが「Mastering Algorithms with C」[注1]では一つの例として演算に優先順位がある計算式の解釈に使う例をあげています．それ以外にもデータベース的な応用や複雑な処理を簡易化するのに役立たせる例もあります．

今まで紹介してきたデータ構造と違い，バイナリ・ツリーの登録，削除には方向，すなわち「左側に対する操作」なのか「右側に対する操作」なのかを区別する必要があります．また，方向の要素を記録するためノード（Node，節）が必要になります．そのため，ノードには最低三つの要素，

1) ノード自身が保持するデータあるいはオブジェクト
2) 左側に連結するノードへの参照
3) 右側に連結するノードへの参照

が必要になります．一方，バイナリ・ツリーには二つの要素，

1) ルート・ノード（みずからがすべての親となる，もっとも根の位置にあるノード）への参照
2) ノードの登録数

が必要になります．

ノードに対する操作として次のようなものが考えられます．

- `data`：ノードに記録しているオブジェクトを返す
- `isLeaf`：ノードがリーフ[注2]であるかを返す
- `left`：左側のノードを返す
- `right`：右側のノードを返す
- `preorder`：ノードの巡回結果（preorder）をコンテナにする
- `inorder`：ノードの巡回結果（inorder）をコンテナにする
- `postorder`：ノードの巡回結果（postorder）をコンテナにする

また，バイナリ・ツリーに対する操作として次のようなものが考えられます．

リスト 1　BiTreeNodeクラスのメンバ変数など

```
template <typename T>
class BiTreeNode {
    friend class BiTree<T>;
public:
    typedef BiTreeNode<T>* NodePtr;

private:
    T mData;          //ノードに記録するオブジェクト
    NodePtr mLeft;   //左側につなぐノード
    NodePtr mRight;  //右側につなぐノード
```

- insertRight：指定したノードの右側に別ノードを追加する
- insertLeft：指定したノードの左側に別ノードを追加する
- removeRight：指定したノードの右側のノードを削除する
- removeLeft：指定したノードの左側のノードを削除する
- root：ルート・ノードを返す
- removeAll：ルート・ノード以下を削除する
- size：バイナリ・ツリーに登録されているオブジェクトの数を返す
- merge：二つのバイナリ・ツリーを合成したバイナリ・ツリーを作成する

バイナリ・ツリーに対する操作のうち，insertRight，insertLeft，removeRight，removeLeftは，よく考えるとツリー全体にではなく，ノードに対する操作ですが，ツリー全体の登録数を変化させるため，あえてバイナリ・ツリーに対する操作に含めています．登録数の取得が不要なら，ノードに対する操作にしてもかまわないと思います．そのように変更したい読者は一度挑戦してみるとよいでしょう．

今回作成したコードはわかりやすいので説明は簡単に済ませます．実際の使用例は後で紹介しますが，それを読めばバイナリ・ツリーの使いかたが理解できるかと思います．

● BiTreeNodeクラスの実装

リスト 1では，ノード(BiTreeNode)とバイナリ・ツリー全体(BiTree)は別クラスに分けています．ここで問題となるのはノードに対するサービスのうち，バイナリ・ツリーから要求されるサービスと，ユーザ(ここではバイナリ・ツリーを利用したいプログラマのこと)から要求されるサービスが微妙に異なっていて，バイナリ・ツリー用サービスをユーザからはアクセス制限したいという点があります．具体的には，ノード変更(changeLeft，changeRight)がそうです．というのも，ノード変更を勝手にされると，バイナリ・ツリーで保持しているノード登録数が正しく変更されなくて困るからです．ユーザからノード変更をする場合にはノード側のサービスではなく，バイナリ・ツリー側のサービスとなります．

アクセス制限はプログラミング言語によって対処が異なる部分ですが，C++の場合にはfriendが使えるでしょう．「friend class BiTree<T>」とすることでバイナリ・ツリー

リスト 2　BiTreeNodeクラスの公開部分

```
public:
    // コンストラクタ
    BiTreeNode(const T& iData) {
        mData = iData;
        mLeft = mRight = NULL;
    }

    // ノードに記録しているオブジェクトを返す
    T data() const {
        return mData;
    }

    //ノードがリーフ(leaf)であるかを返す
    bool isLeaf() const {
        return mLeft == NULL && mRight == NULL;
    }

    //左側ノードを返す
    NodePtr left() const {
        return mLeft;
    }

    //右側ノードを返す
    NodePtr right() const {
        return mRight;
    }

    //ノードの巡回結果(preorder)を得る
    void preorder(std::deque<NodePtr>& oDeque) {
        preorder(this,oDeque);
    }

    //ノードの巡回結果(inorder)を得る
    void inorder(std::deque<NodePtr>& oDeque) {
        inorder(this,oDeque);
    }

    //ノードの巡回結果(postorder)を得る
    void postorder(std::deque<NodePtr>& oDeque) {
        postorder(this,oDeque);
    }
```

(BiTree)からはノードの非公開メンバを自由にアクセスできるようになります．friendのないプログラミング言語，たとえばJavaだったらノード・クラスをバイナリ・ツリー・クラスのインナ・クラスにすることでアクセス制限ができるでしょう[注3]．

注意してほしいのは，friendは悪用されるとどうしようもない弊害をもたらす危険なしくみだという点です．オブジェクト指向の考えでは外部に対して実装の詳細を隠すことで将来的な仕様変更を楽にしたり，外部から利用する場合に知るべき情報を絞り込むことで楽に取り扱わせる意図があるのですが，それらに真っ向から逆らう危険性があります．実際のところ，筆者がfriendを使うのは，ここで示したような互いの関連性や結びつきの強いクラスや演算子のオーバロードなどでやむを得ない状況にのみ限定しています．

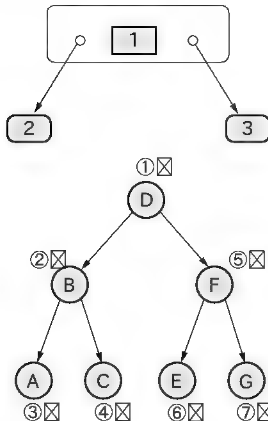
リスト 2がBiTreeNodeクラスの公開部分です．このあたりは難しくないので説明を省略します．ただ，ノードの巡回結果を得る関数はわかりにくいので補足説明します．

バイナリ・ツリーのノードを巡回するとき，

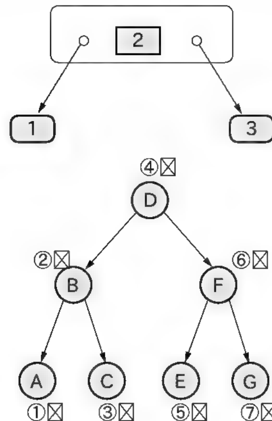
- preorder：自ノード→左側ノード→右側ノードをたどる
- inorder：左側ノード→自ノード→右側ノードをたどる

注3：最初に作成したJava版ではインナ・クラスにしている．Java版は本誌8月号のCD-ROMに収録する予定．

preorderは自ノード→左側ノード→右側ノードをたどる



inorderは左側ノード→自ノード→右側ノードをたどる



postorderは左側ノード→右側ノード→自ノードをたどる

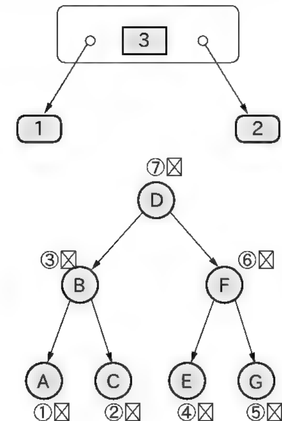


図2　バイナリ・ツリーの巡回

リスト 3　BiTreeNodeクラスの非公開部分

```
//左側ノードを変更する
void changeLeft(NodePtr iNode) {
    mLeft = iNode;
}

//右側ノードを変更する
void changeRight(NodePtr iNode) {
    mRight = iNode;
}

void preorder(NodePtr iNode,std::deque<NodePtr>& oDeque) {
    if(iNode != NULL){
        oDeque.push_back(iNode);
        preorder(iNode->left(),oDeque);
        preorder(iNode->right(),oDeque);
    }
}

void inorder(NodePtr iNode,std::deque<NodePtr>& oDeque) {
    if(iNode != NULL){
        inorder(iNode->left(),oDeque);
        oDeque.push_back(iNode);
        inorder(iNode->right(),oDeque);
    }
}

void postorder(NodePtr iNode,std::deque<NodePtr>& oDeque) {
    if(iNode != NULL){
        postorder(iNode->left(),oDeque);
        postorder(iNode->right(),oDeque);
        oDeque.push_back(iNode);
    }
}
```

リスト 4　BiTreeクラスのメンバ変数など

```
template <typename T>
class BiTree {
public:
    typedef BiTreeNode<T>  Node;
    typedef BiTreeNode<T>* NodePtr;

private:
    NodePtr mRoot; //ルート・ノード
    int mSize;     //オブジェクトの登録数

public:
    // コンストラクタ
    BiTree() {
        mRoot = NULL;
        mSize = 0;
    }

    // デストラクタ
    ~BiTree(){
        removeAll(); // ルート・ノード以下を削除する
    }

    // 登録数を返す
    int size() const {
        return mSize;
    }

    // ルート・ノードを返す
    NodePtr root() const {
        return mRoot;
    }
```

●postorder：左側ノード→右側ノード→自ノードをたどる

の3パターンが考えられます(図2)[注4]．バイナリ・ツリーがソートずみの状態，たとえば左側ノードにあるオブジェクトが自ノードにあるオブジェクトより小さい値で，右側ノードにあるオブジェクトが自ノードにあるオブジェクトより大きい値になっている状態では，inorderで巡回することはソートされた順番でノードを巡回することと同じです．

リスト3がBiTreeNodeクラスの非公開部分です．巡回結果をえる関数は再帰を上手に利用することで案外簡単に実装できます．

● BiTreeクラスの実装

次はバイナリ・ツリーの本体を実現するクラス(BiTree)です．

リスト4がBiTreeクラスのメンバ変数，リスト5がノードを追加するメンバ関数です．左右ごとに用意されていますが，方向が違うだけで処理内容はほとんど同じようなものです．

注4：「Mastering Algorithms with C」ではlevel orderというパターンも紹介している．自ノードから開始して子ノードをたどり，次に孫ノード，その次は曾孫ノードをたどるというパターン．

リスト 5　追加のメンバ関数

```
// 指定ノードの右側にノードを追加する
// iNodeがNULLかつルート・ノードがないなら追加ノードは
//  ルート・ノードになる
// iNode 指定ノード
// iData 追加ノードに登録するオブジェクト
// return 追加できたなら作成されたnode，できなかったならNULL
NodePtr insertRight(NodePtr iNode,const T& iData){
    NodePtr aNode = NULL;
    if(iNode == NULL){
        if(mSize == 0){
            mRoot = aNode = new Node(iData);
        }
    }else{
        if(iNode->right() == NULL){
            iNode->changeRight(aNode = new Node(iData));
        }
    }
    if(aNode != NULL){
        ++mSize;
    }
    return aNode;
}

// 指定ノードの左側にノードを追加する
// iNodeがNULLかつルート・ノードがないなら追加ノードは
//  ルート・ノードになる
// iNode 指定ノード
// iData 追加ノードに登録するオブジェクト
// return 追加できたなら作成されたnode，できなかったならNULL
NodePtr insertLeft(NodePtr iNode,const T& iData){
    NodePtr aNode = NULL;
    if(iNode == NULL){
        if(mSize == 0){
            mRoot = aNode = new Node(iData);
        }
    }else{
        if(iNode->left() == NULL){
            iNode->changeLeft(aNode = new Node(iData));
        }
    }
    if(aNode != NULL){
        ++mSize;
    }
    return aNode;
}
```

リスト 6はノードを削除するメンバ関数です．removeSubは指定ノード，およびそのノードにつながる子ノードを再帰的に削除します．リスト 7は二つのバイナリ・ツリーを合成したツリーを作るメンバ関数です．

● バイナリ・ツリーの利用例

バイナリ・ツリーを使った例として，登録したデータを内部でソート済みにし，取り出すときはソート済みのコンテナ(ここではdeque)で取り出すようなキュー構造を実装してみます．

リスト 8ではpushでデータ登録，popでデータを取り出します．pushのとき登録データをルート・ノードにあるデータと比較し，それよりも小さいなら左側のノードに追加できるかを検討し，大きいか等しいなら右側のノードに追加できるかを検討します．すでに左右のノードが存在するなら，登録データをそのノードと比較する作業を再帰的に行います．そうすることであらかじめソートされた順を保ったバイナリ・ツリーが形成されます(図 3)．取り出すときはinorderでツリーを巡回するとソート済みでデータを取り出せるわけです．sorterを使ったサンプルはリスト 9のようになります．

リスト 6　削除のメンバ関数

```
    // ルート・ノード以下を削除する
    void removeAll(){
        removeSub(mRoot);
        mRoot = NULL;
        mSize = 0;
    }

    // 指定ノードの右側を削除する
    // iNodeがNULLかつルート・ノードがあるなら
    //  ルート・ノードを削除する
    // iNode 指定ノード
    void removeRight(NodePtr iNode){
        if(mSize > 0){
            if(iNode == NULL){
                removeAll();
            }else{
                if(removeSub(iNode->right())){
                    iNode->changeRight(NULL);
                }
            }
        }
    }

    // 指定ノードの左側を削除する
    // iNodeがnullかつルート・ノードがあるなら
    //  ルート・ノードを削除する
    // iNode 指定ノード
    void removeLeft(NodePtr iNode){
        if(mSize > 0){
            if(iNode == NULL){
                removeAll();
            }else{
                if(removeSub(iNode->left())){
                    iNode->changeLeft(NULL);
                }
            }
        }
    }

private:
    bool removeSub(NodePtr iNode){
        if(iNode != NULL){
            removeLeft(iNode);
            removeRight(iNode);
            delete iNode;
            --mSize;
            return true;
        }else{
            return false;
        }
    }
```

リスト 7　merge

```
// 二つのバイナリ・ツリーを合成したバイナリ・ツリーを
//  作成する
// 合成後，二つのバイナリ・ツリーは空にされる
// iLeft 左側にするバイナリ・ツリー
// iRight 右側にするバイナリ・ツリー
// iData 新規作成されるバイナリ・ツリーのルート・ノードに
//  登録するオブジェクト
void merge(BiTree<T>& iLeft,BiTree<T>& iRight,const T&
                                                  iData){

    //自分を空にする
    removeAll();

    //ルート・ノードにオブジェクトを登録する
    insertLeft(NULL,iData);

    //ルート・ノードの左右にiLeft,iRightの
    // ルート・ノードをつなぐ mRoot->changeLeft(iLeft.root());
    mRoot->changeRight(iRight.root());
    mSize += (iLeft.size() + iRight.size());

    //iLeft,iRightを空にする
    iLeft.mRoot = NULL;
    iLeft.mSize = 0;
    iRight.mRoot = NULL;
    iRight.mSize = 0;
}
```

リスト 8　sorterクラス

```
template <typename T>
class sorter {
//private:
    BiTree<T> mBiTree;
public:
    void push(const T& iObj){ //データの登録
        //最初のオブジェクトならルート・ノードにする
        BiTreeNode<T>* aNode = mBiTree.root();
        if(aNode == NULL){
            mBiTree.insertRight(NULL,iObj);
            return;
        }
        for(;;){
            //ノードにあるオブジェクトと
            // pushしたオブジェクトを比較する
            if(iObj < aNode->data()){
                //pushしたほうが小さい場合
                //ツリーの左側が空いているなら,
                // そこにノードを新規作成する
                if(mBiTree.insertLeft(aNode,iObj) != NULL){
                    return;
                }
                //作成できないなら，左側にあるノードで
                // 再検討する
                aNode = aNode->left();
            }else{
                //pushしたほうが大きいか等しい場合
                //ツリーの右側が空いているなら,
                // そこにノードを新規作成する
                if(mBiTree.insertRight(aNode,iObj) != NULL){
                    return;
                }
                //作成できないなら，右側にあるノードで
                // 再検討する
                aNode = aNode->right();
            }
        }
    }

    void pop(std::deque<T>& oDeque) const { //データの取り出し
        BiTreeNode<T>* aRootNode = mBiTree.root();
        if(aRootNode != NULL){
            std::deque<BiTreeNode<T>* > aNodeDeque;
            aRootNode->inorder(aNodeDeque);
            while(!aNodeDeque.empty()){
                const BiTreeNode<T>* aObj =aNodeDeque.front();
                oDeque.push_back(aObj->data());
                aNodeDeque.pop_front();
            }
        }
    }
};
```

リスト 9　sorterの利用例

```
static void demo5()
{
    std::cout << "* demo5 *¥n";

    sorter<std::string> aSorter;

    aSorter.push(std::string("[20]"));
    aSorter.push(std::string("[09]"));
    aSorter.push(std::string("[53]"));
    aSorter.push(std::string("[05]"));
    aSorter.push(std::string("[15]"));
    aSorter.push(std::string("[44]"));
    aSorter.push(std::string("[79]"));
    aSorter.push(std::string("[01]"));
    aSorter.push(std::string("[07]"));
    aSorter.push(std::string("[11]"));
    aSorter.push(std::string("[01]"));
    aSorter.push(std::string("[20]"));
    std::deque<std::string> aDeque;
    aSorter.pop(aDeque);
    dumpDeque<std::string>("aDeque = ",aDeque);
}

template <typename T>
static void dumpDeque(const char* iText,
                      const std::deque<T>& iDeque)
{
    typedef std::deque<T> Deque;

    std::cout << iText << "(size=" << iDeque.size() << ")";
    typename Deque::const_iterator aItr;
    for(aItr = iDeque.begin(); aItr != iDeque.end(); aItr++){
        std::cout << " " << *aItr;
    }
    std::cout << std::endl;
}
```

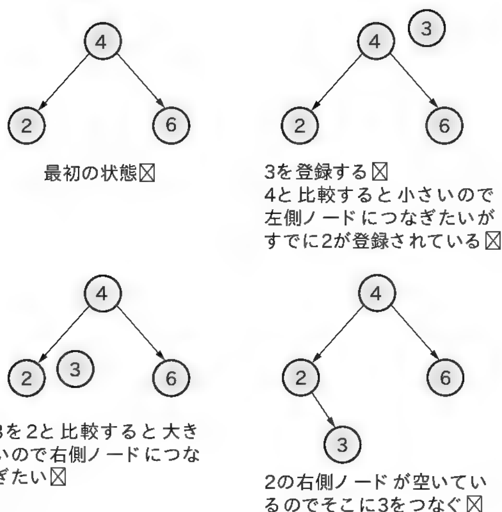
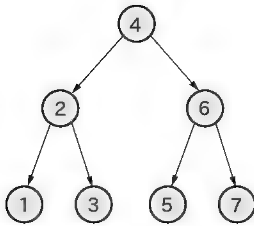


図3　バイナリ・ツリーを使ったソート

ヒープ(Heap)

バイナリ・ツリーを使ってソートするときに問題となるのは，あらかじめソート済みの順番でデータが登録されるとツリーのバランスが偏ってしまい，かえって実行効率が悪くなることです(図4)．そのためバイナリ・ツリーのバランスを自動的に保ってくれるAVLツリーや赤黒木，あるいは二分木ではなく多分木を使ったB木などが考案されています．しかし，いずれも実装がめんどうなことから気軽にプログラミングしにくいので，それらを自分で一から実装することはあまりありません．たいていはできあいのライブラリを利用することになるでしょう．

とはいうものの，優先度の大きい順にデータを取り出す優先順位付きキューを実装したい局面は多いことでしょう．そのときに，できあいのライブラリが流用しにくい場合もあります．簡単に実装するなら連結リストを用意して挿入ソートを行えば



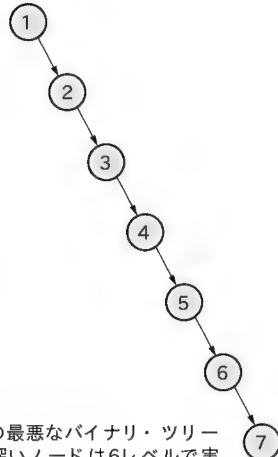


図4　バイナリ・ツリーのバランス

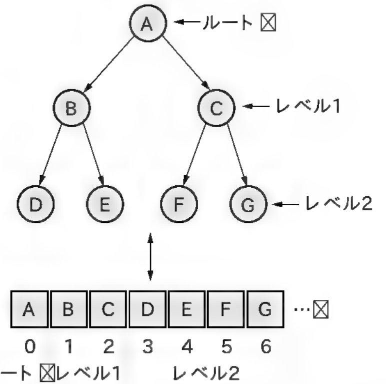


図5　ヒープ

良いのですが，登録数が多いと速度的に不利になります．だからといって，固定配列に毎回クイック・ソートをほどこすのもおおげさです．

そこで紹介したいのは，バランスのとれたバイナリ・ツリーを固定配列に対応させた「ヒープ」というデータ構造です（図5）．ヒープということばはフリー・メモリを確保する領域を指すこともありますが，ここで紹介するのはそういった領域の意味ではないことに注意してください．ヒープの正体は拡張可能な配列です．ただし，あるノードが対応する配列のインデックスは順番通りではなく，以下のような規則で対応します．

1）配列の0番目（最初のインデックス）をルート・ノードとする（レベル0のノード）
2）配列の1番目をルート・ノードの左側子ノードとする（レベル1のノード）
3）配列の2番目をルート・ノードの右側子ノードとする（レベル1のノード）
4）配列の3番目を2）のノードの左側子ノードとする（レベル2のノード）
5）配列の4番目を2）のノードの右側子ノードとする（レベル2のノード）
6）配列の5番目を3）のノードの左側子ノードとする（レベル2のノード）
7）配列の6番目を3）のノードの右側子ノードとする（レベル2のノード）
…（以下続く）…

ルート・ノードからの距離をnレベルと考えると，ルート・ノード以外に関しては，

- レベルnのノードは（n×2）個

となります．

あるノードが対応するインデックスがiだとすれば，

- あるノードの左側子ノードのインデックスは（i×2）+1
- あるノードの右側子ノードのインデックスは（i×2）+2

となります．

ルート・ノード以外のノードでは，

- あるノードの自ノードのインデックスは（i−1）÷2

となります．

また，レベルが大きいノードのデータはレベルが小さいノードのデータよりも小さくなっています．ルート・ノードにあるデータが最大であり，レベルが大きくなるにつれてデータはどんどん小さくなっていくという前提です．

以上の規則がわかればヒープを利用したバイナリ・ツリーの実装はさほど難しくありません．リスト10にこの実装を示します．

リスト 10　Heapクラスのメンバ変数など

```
template <typename T>
class Heap {
//private:
    std::vector<T> mTree; //オブジェクトを格納するリスト
    unsigned int mSize;   //有効なオブジェクトの数

    //自ノードのインデックスをえる
    unsigned int parentPos(unsigned int iPos) const {
        return (iPos - 1) / 2;
    }
    //左側の子ノードのインデックスをえる
    unsigned int leftPos(unsigned int iPos) const {
        return iPos * 2 + 1;
    }
    //右側の子ノードのインデックスをえる
    unsigned int rightPos(unsigned int iPos) const {
        return iPos * 2 + 2;
    }
public:
    // コンストラクタ
    Heap(){
        mSize = 0;
    }

    // 登録数を返す
    unsigned int size() const {
        return mSize;
    }
```

リスト 11　Heapクラスの登録メンバ関数

```
// オブジェクトをヒープに登録する
void enqueue(const T& iObj) {

    //格納リストの最後尾にオブジェクトを置く
    if(mTree.size() > mSize){
        mTree[mSize] = iObj;
    }else{
        mTree.push_back(iObj);
    }

    //今置いたオブジェクトとすでにリストにある
    // オブジェクト同士の順位を保つように交換する
    unsigned int aIpos = mSize;
    unsigned int aPpos = parentPos(aIpos);
    while(aIpos > 0){
        if(mTree[aIpos] > mTree[aPpos]){
            std::swap(mTree[aIpos],mTree[aPpos]);
            aIpos = aPpos;
            aPpos = parentPos(aIpos);
        }else{
            aIpos = 0;
        }
    }
    //格納数が増える
    ++mSize;
}
```

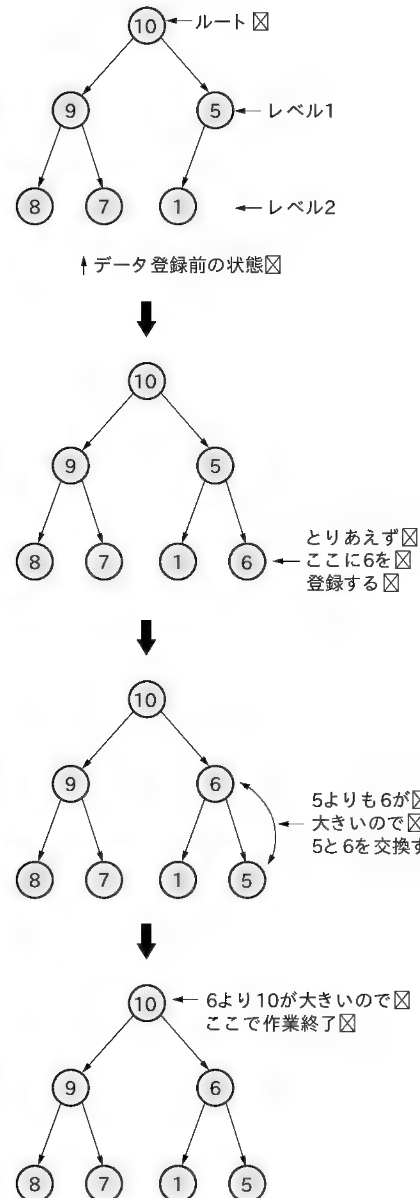


図6　ヒープへのデータ登録

自動的にサイズを拡張する配列(mTree)はC言語ではreallocを使って実装するところですが，C++ならSTLのコンテナ(vector，deque)を使うほうが便利でしょう．ここではvectorを使います．

● ヒープへのデータ登録

ヒープにデータを登録するときは配列の最後尾にデータを置きます．しかし，このままではツリー上の大小関係が保たれているかどうかが不明です．大小関係を保つように，登録データとすでにヒープに登録したデータを比較して交換する作業を行います(図6)．その結果，登録データが大きいならばどんどん浮上していきます(リスト11)．

● ヒープからのデータ取り出し

ヒープから最大値を取り出すのは単に配列の最初を取り出すだけです．しかし取り出した後，抜けた穴を埋める処理が必要です．この処理ではおもしろいことに配列の最後尾を取り出し，それを抜けた穴に置きます．当然のことながら最後尾にあったデータが最大値であるはずはなく，ここからデータを沈めていく処理を行います(図7)．単に子ノードと大きさを比較するだけでなく，左右のどちらに沈めていくかも決めないといけないので，少々ややこしいかもしれません．リスト12に例を示します．

みやさか・でんと　miyadent@anet.ne.jp

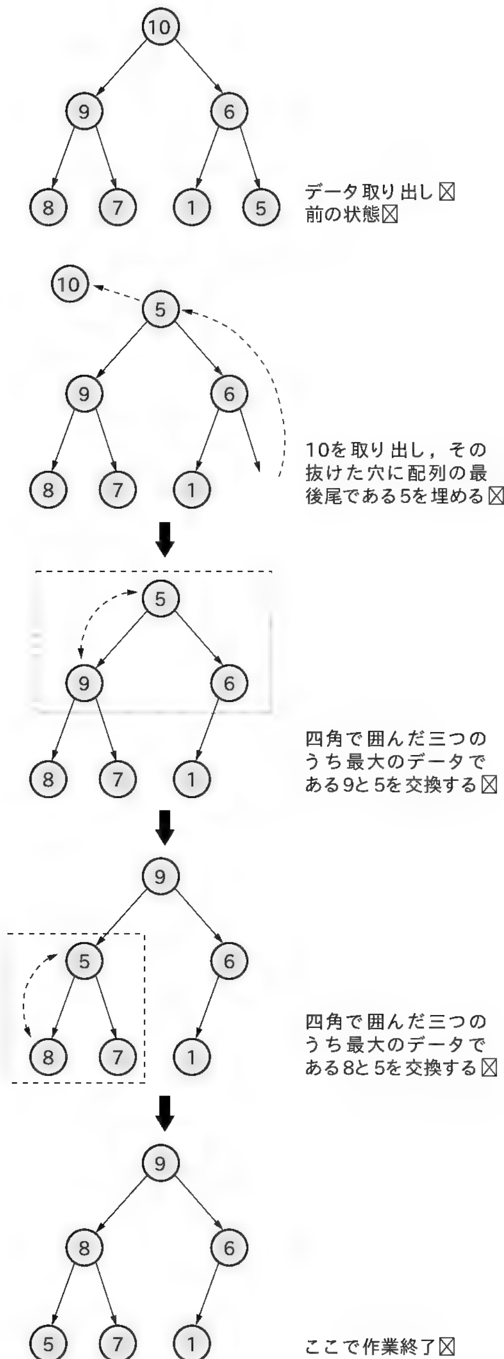


図7　ヒープからのデータ取り出し

リスト 12　Heapクラスの取り出しメンバ関数

```
// ヒープの先頭にあるオブジェクトがあるなら，
// それをoObjに代入してtrueで戻る
// ないならfalseで戻る
bool peek(T& oObj) const {
    if(mSize == 0){
        return false;
    }else{
        oObj = mTree[0];
        return true;
    }
}

// ヒープの先頭のオブジェクトを取り出しoObjに代入する
// 成功すればtrue，そうでないならfalseで戻る
bool dequeue(T& oObj) {
    //空っぽなら戻る
    if(mSize == 0){
        return false;
    }

    //先頭にあるオブジェクトを取り出してoObjに代入する
    oObj = mTree[0];

    //格納数を減らす
    if(--mSize == 0){
        return true;
    }

    //最後尾のオブジェクトを先頭にする
    mTree[0] = mTree[mSize];

    //先頭のオブジェクトとすでにリストにある
    // オブジェクト同士を順位を保つように交換する
    unsigned int aIpos = 0;
    unsigned int aMpos;
    do{
        unsigned int aRpos = rightPos(aIpos);
        unsigned int aLpos = leftPos(aIpos);

        //ノードのオブジェクトと左側のオブジェクトと比較する
        if(aLpos < mSize && mTree[aLpos] > mTree[aIpos]){
            aMpos = aLpos;
        }else{
            aMpos = aIpos;
        }

        //右側のオブジェクトと比較する
        if(aRpos < mSize && mTree[aRpos] > mTree[aMpos]){
            aMpos = aRpos;
        }

        //オブジェクトを交換すべきなら交換する
        if(aMpos == aIpos){
            aIpos = 0;
        }else{
            std::swap(mTree[aIpos],mTree[aMpos]);
            aIpos = aMpos;
        }
    }while(aIpos > 0);
    return true;
}
```

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第㉖回 アセンブラを使いこなすための基礎知識とC言語からのアセンブラの使用方法(MASM編:その1)

大貫 広幸

これまで本連載では，アセンブラMASMやgasの基本的な操作方法や，x86系32ビットCPUの命令セットを説明するために最低限必要と思われるソース・ファイルの構成については説明してきました．しかし，実際にアセンブラでプログラムを作成する場合，まだまだMASMやgasの知識が必要となります．

WindowsやLinuxのプログラムを作成する場合，CやC++といった高級言語を使うことが一般的で，アセンブラですべてをプログラミングすることはなく，高級言語から呼び出されるサブルーチンとしてアセンブラを使用する場合がほとんどです．

そこで，ここでは目標をC(C++)から呼び出されるアセンブラのサブルーチンを作成する場合に絞り，そのために必要な最低限の事がらについて解説します．

また，C(C++)のソース・プログラム中に，直接アセンブラを記述するためのインライン・アセンブラについても説明します．

アセンブラMASMを使いこなすために最低限必要な知識

ここでは，アセンブラMASMでプログラミングする場合，筆者が最低これだけは覚えておいたほうがよいと考えていることについて説明します．

● アセンブラとアドレス

アセンブラを使用する場合，変数やサブルーチンがメモリ上のどのアドレスに配置されるかといったことや，そのアドレスをどのようにプログラムで扱うかといったことも知っておく必要があります．ここでは，このようなアセンブラとアドレスの関係について説明します．

(1) セグメントとロケーション・カウンタ

ロケーション・カウンタとは，アセンブラがデータや機械語コードをセグメントに配置するときに使用するカウンタで，その値はソース・ファイル単位のモジュール内でのみ有効です．

ロケーション・カウンタは，セグメントごとにあり，初期値はゼロになっています．ロケーション・カウンタは，次に生成されるデータや機械語コードを配置するセグメント(セグメントの最初のバイトを0とする)の相対アドレス(論理セグメントのオフセット)を示しています．ロケーション・カウンタは，データや機械語コードをセグメント内に配置すると，そのバイト数分だけプラスされます．つまり，今まで説明で使用してきた，MASMのプログラムの`.data`や`.data?`，そして`.code`の各セグメントは，すべて独自のロケーション・カウンタを持つわけです(図1)．

現在のロケーション・カウンタの値は「`$`」により取得できます．この`$`は定義済みシンボルとしてMASM自身が定義しているシンボルで，使い方は通常のラベル参照と同じです．ただし，`$`は現在のロケーション・カウンタの値なので，参照するたびにその値は異なります．

ロケーション・カウンタの値を変更するディレクティブには，以前この連載で説明した`.ALIGN`のほかに`EVEN`や`ORG`があります．`EVEN`は，`EVEN`の次の命令やデータを偶数アドレスから配置するものです．そして，`ORG`はオペランドで指定されたアドレスにロケーション・カウンタを強制的にセットするディレクティブです．リスト1は，このロケーション・カウンタについての説明を実際のリストで示したものです．

(2) 定数式とアドレス

整数の定数式については以前この連載で説明しましたが，アドレスが定数式に使われた場合については，まだ説明していませんでした．

シンボルのアドレスは，アセンブル段階では論理的セグメント上のオフセットとして存在します．最終的な実行時のアドレスは，プログラムをメモリにロードするローダにより決定します．そのため，アドレスに対する演算というのは，ローダの機能ということになります(図2)．ローダの演算機能は限られていて，アドレスに整数の定数を加減算する程度の機能しかありません．

このことから，定数式にアドレスが使われた場合というのは，アドレスを格納しているシンボルを`SYM`とした場合,「`SYM`」のみか「`SYM`±結果が整数となる定数式」のいずれかになります．この場合の定数式はアドレスを結果として返すことになります(リスト2)．

また，モジュール内で定義されたシンボルどうしなら減算も使用できます．この場合，減算の結果は整数になります．たとえば，`SYM1`と`SYM2`というモジュール内で定義されたシンボ

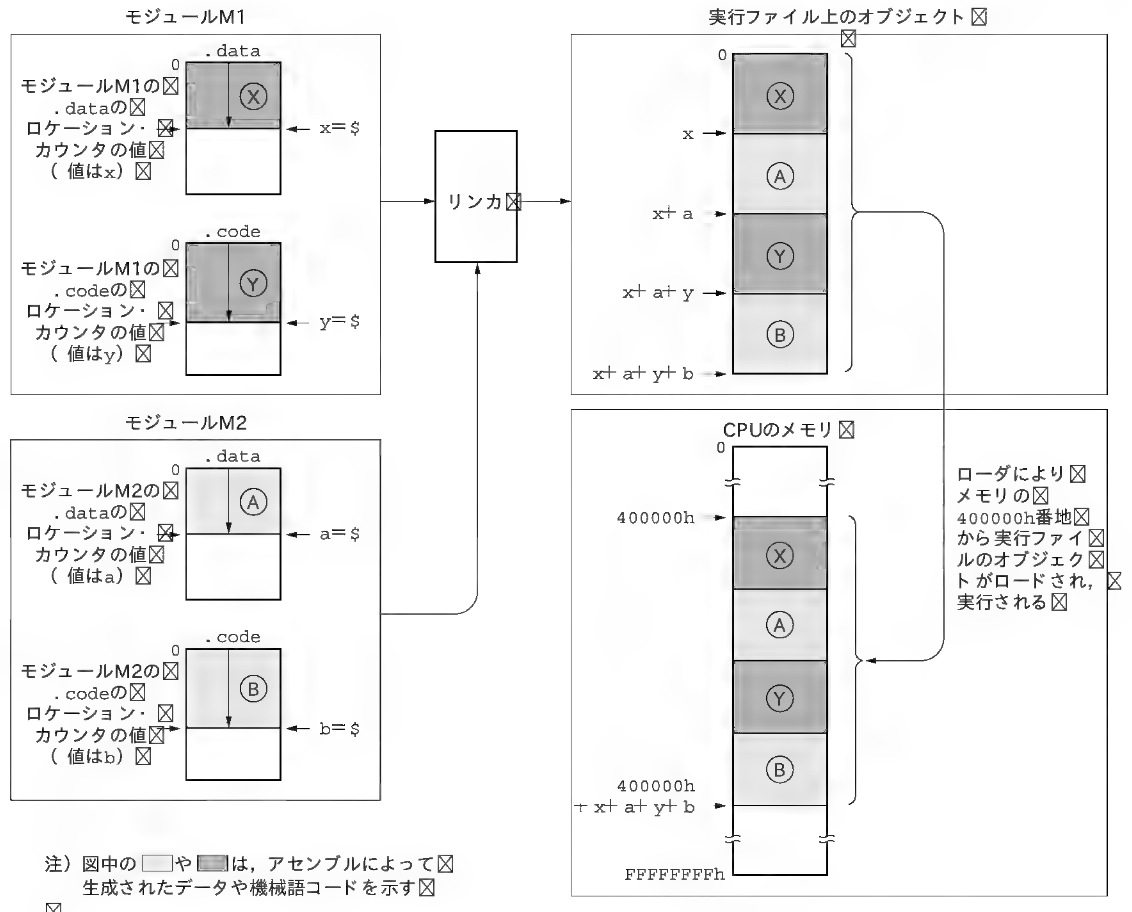
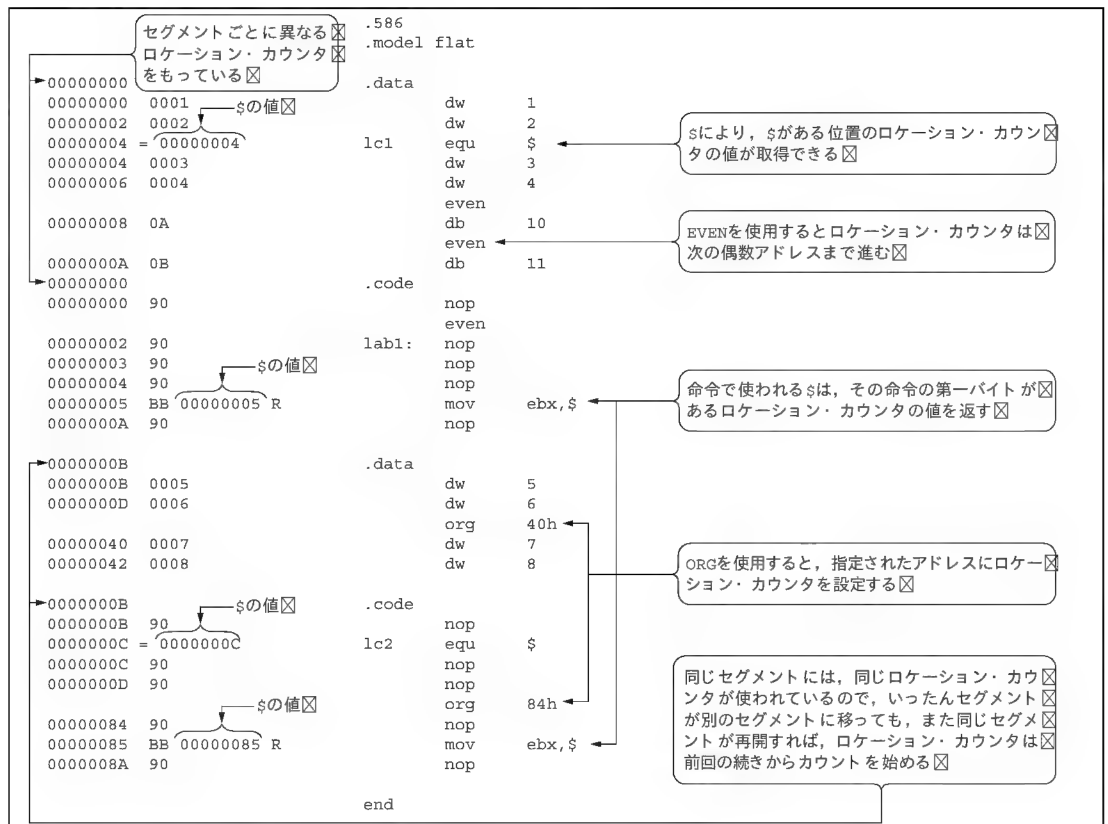


図1
ロケーション・カウンタ

リスト1
MASMのロケーション・カウンタ（$，EVEN，ORG）



```
                              .586
                              .model flat
00000000                      .data
00000000  0001                       dw    1
00000002  0002                       dw    2
00000004 = 00000004           lc1    equ   $
00000004  0003                       dw    3
00000006  0004                       dw    4
                                     even
00000008  0A                         db    10
                                     even
0000000A  0B                         db    11
00000000                      .code
00000000  90                         nop
                                     even
00000002  90                  lab1:  nop
00000003  90                         nop
00000004  90                         nop
00000005  BB 00000005 R              mov   ebx,$
0000000A  90                         nop

0000000B                      .data
0000000B  0005                       dw    5
0000000D  0006                       dw    6
                                     org   40h
00000040  0007                       dw    7
00000042  0008                       dw    8

0000000B                      .code
0000000B  90                         nop
0000000C = 0000000C           lc2    equ   $
0000000C  90                         nop
0000000D  90                         nop
                                     org   84h
00000084  90                         nop
00000085  BB 00000085 R              mov   ebx,$
0000008A  90                         nop

                              end
```

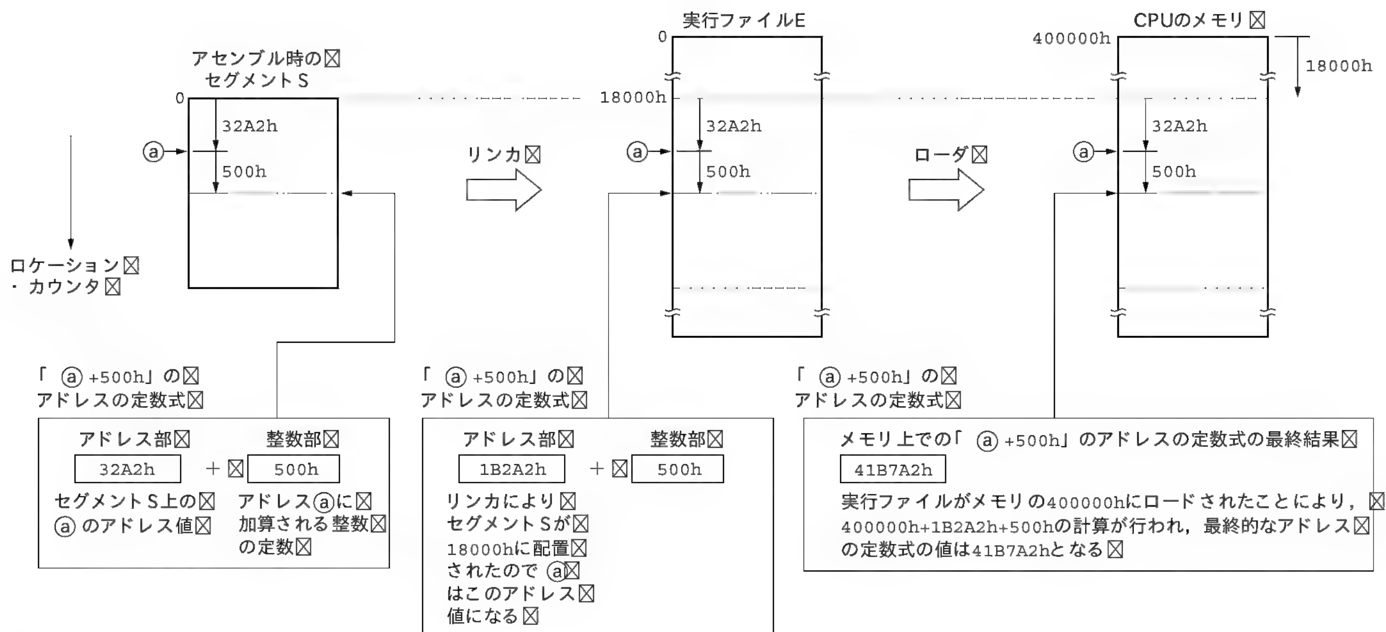


図2　ローダによるアドレス計算

リスト 2　MASMの定数式とアドレス

```
 00000004 01 02 03 04 05        va      db      1,2,3,4,5,6
           06
 0000000A 0002 0003 0004        vb      dw      2,3,4,5
          0005
 00000012 00000004              vc      dd      4,5,6,7,8,9,10,11
          00000005
          00000006
          00000007
          00000008
          00000009
          0000000A
          0000000B
 00000032 = 00000032            vd      equ     $

 00000032  00000004 R                   dd      va,vb,vc,vd
           0000000A R
           00000012 R
           00000032 R
 00000042  0000000C R                   dd      va+8,vb-4
           00000006 R

 0000004A  0010                         dw      (vc-vb)*2

                                        dw      (vc+vb)*2
ap26_2_5.asm(18) : error A2101: cannot add two relocatable labels


 00000003  66| A1                       mov     ax,vb
           0000000A R
 00000009  66| A1                       mov     ax,vb+12h
           0000001C R
 0000000F  66| A1                       mov     ax,vb-8
           00000002 R
                                        mov     ax,vb+vd
ap26_2_5.asm(28) : error A2101: cannot add two relocatable labels

 00000015  B8 00000006                  mov     eax,vb-va
 0000001A  B8 00000028                  mov     eax,vd-vb
 0000001F  B8 00000004                  mov     eax,(vc-vb)/2
 00000024  B8 00000008                  mov     eax,(vd-vc)/4
```

「(アドレス)±(整数値)=(アドレス)」となる

減算以外のアドレスどうしの計算はエラーとなり，できない

「(アドレス)−(アドレス)=(整数値)」となる

ルなら「SYM1 − SYM2」の減算のみで，SYM1からSYM2までのバイト数を整数として得ることができます(リスト2)．

● シンボルと定数式

以前，MASMのシンボルは属性をもっていると述べました．そのため，オペランドにシンボルを記述するだけで，シンボルがデータなのかラベルなのか，データならどんな型で何個定義されているかといったことがわかるため，MASMはその属性に合ったコードを生成してくれます．

```
                        +3 +2 +1 +0
 var DWORD 12345678h ……☒ 12 34 56 78

▶ダブルワードの値すべてをアクセスする場合☒
☒  MOV   EXA, Var…………OK，レジスタ EAX には 12345678h が入る
▶ダブルワードの値一部をアクセスした場合☒
☒  MOV   AL, var+1……… NG，var はダブルワードなので，レジスタ
                          AL ではアクセスできない．
   MOV   AL, BYTE PTR var+1……OK,「BYTE PTR」により，var は☒
                                バイトのデータとなるため，
                                レジスタ AL によるアクセスが可能☒
                                になる．実行するとレジスタ AL に☒
                                は 56h が入る☒
```

（a）PTRの使用例1（シンボルの型をPTRで変更する場合）☒
☒

```
▶レジスタ EBX が示すメモリ上のワード値を＋1する☒
☒  INC ［EBX］……………………NG，レジスタ EBX が示すメモリ上の☒
                              値の型がわからない☒
   INC  WORD PTR ［EBX］……OK,「WORD PTR」によってレジスタ☒
                              EBX が示すメモリ上の値をワードと指☒
                              定したので，ワード値に対する INC 命☒
                              令が生成される☒
```

（b）PTRの使用例2（必ずPTRを使う必要がある場合）☒

リスト3
MASMの **PTR** 演算子と使用例

```
 0000004E 12345678            varDD1  dd      12345678h
 00000052 3F800000            r4pak   real4   1.0,2.0,3.0,4.0
          40000000
          40400000
          40800000

                                      inc     [ebx] ←
ap26_2_5.asm(42) : error A2023: instruction operand must have size
 00000034  66| FF 03                  inc     word ptr [ebx]  }←
 00000037  FF 03                      inc     dword ptr [ebx] }

 00000039  A1 0000004E R              mov     eax,varDD1
                                      mov     al,varDD1+1      }←
ap26_2_5.asm(47) : error A2070: invalid instruction operands   }
 0000003E  A0 0000004F R              mov     al,byte ptr varDD1+1 }
 00000043  66| A1                     mov     ax,word ptr varDD1   }←
           0000004E R                                              }
 00000049  66| 8B 15                  mov     dx,word ptr varDD1+2 }
           00000050 R

 00000050  0F 10 03                   movups  xmm0,[ebx]
 00000053  0F 10 03                   movups  xmm0,oword ptr [ebx] ←
 00000056  0F 10 0D                   movups  xmm1,r4pak
           00000052 R
 0000005D  0F 10 0D                   movups  xmm1,oword ptr r4pak ←
           00000052 R
```

［EBX］のままだと，アクセスするメモリ☒上のデータ・サイズがわからないので☒エラーとなる☒

PTR演算子で型を指定すると，上記のような☒エラーは発生しなくなる☒

シンボル定義時のデータ・サイズ☒（型）以外で，メモリをアクセスする☒のもエラーとなる☒

上記のような場合にも☒PTR演算子を使用する☒

メモリ上の128ビット（16バイト）データ☒をアクセスする場合は"OWORD PTR"を☒使用する．"OWORD"のデータ・タイプ☒は，PTR演算子でのみ使用可能．データ☒の定義には使用できない☒

しかし，これではシンボルを別の属性で参照したい場合に困ります．そこで，MASMにはシンボルを別の属性に変更する機能もあります．また，プログラムによっては，シンボルがもつアドレスや型のバイト数，データの個数，シンボル全体のバイト数といった情報も必要な場合もあります．

ここでは，これらシンボルについての事がらを説明します．

（1）シンボルの型の変更（**PTR** 演算子）

シンボルの型の変更は PTR 演算子で行います．PTR 演算子は，左辺に設定したいタイプ（型），右辺に結果がアドレスとなる定数式を記述することで，指定された定数式が指定タイプに変更されます．また，オペランドのみでは型が特定できない場合も，この PTR 演算子を使用する必要があります（図3）．

タイプとしては，データの場合，定義で使用するディレクティブがタイプとなります．ただし，Dで始まる2文字のディレクティブは使用できません（リスト3）．ラベルの場合は，NEAR と FAR が指定できますが，Windowsの32ビット・プログラムでは，NEAR のみが使われるので，ラベルに対する PTR 演算子は通常，使用されません．

（2）変数やラベルのアドレスの取得（**OFFSET** 演算子）

プログラムを作成していると，変数やラベルのアドレスをどうしても取得したい場合が発生します．この場合，方法としてはCPUの LEA 命令を使う方法と「OFFSET」の演算子を使う方法の2種類があります．LEA 命令はすでにCPU命令の回で説明済みです．

OFFSET 演算子は単項演算子で，結果がアドレスとなる定数式の前に記述することで，定数式のアドレスが整数値として取得できます．この OFFSET 演算子で得られた整数値となったアドレスは，データ定義時の初期値やCPU命令のイミディエイトのオペランドに使用できます．

ただし，OFFSET 演算子でアドレスが整数値となるわけですが，上記「アセンブラとアドレス」の「（2）定数式とアドレス」の理由から OFFSET 演算子で得られた整数値に対する演算は，ローダが行うため，アドレスを含まない定数式との加減算のみとなります．

また，OFFSET 演算子を使用する場合の注意として，MASMのバージョンによって単に「OFFSET 結果がアドレスとなる定

数式」の指定では，実行時のアドレスが正しく得られない場合があります．そのため，32ビット・プログラムでこのOFFSETを使用する場合は，「OFFSET FLAT：結果がアドレスとなる定数式」と指定します（リスト4）．

これは，プログラム上セグメントは，.data，.data?，.codeに分かれていますが，「.model flat」の指定により.data，.data?，.codeのセグメントがグループ化され，一つのセグメントとして扱われます．

MASMのバージョンによっては，OFFSETはソース上のシンボルが定義されているセグメント上のオフセットを返す場合があります．しかし，これではグループ化されたセグメント上のオフセットを取得することができません．また，通常必要なのはこのグループ化されたセグメント上のオフセットなのです．

そこで，グループ化されたセグメント上のオフセットを取得する場合は，結果がアドレスとなる定数式の前に，グループに付けられた名前「FLAT」をセグメント名として指定します（図4）．

リスト4　MASMの**OFFSET**演算子，**TYPE**演算子，**LENGTH**演算子，**SIZE**演算子の使用例

```
00000062 00 00              a0001   byte    ?,?
00000064 0000 0000          a0002   word    ?,?
00000068 00000000           a0003   dword   ?,?
         00000000
00000070                    a0004   qword   ?,?
          0000000000000000
          0000000000000000
00000080                    a0005   tbyte   ?,?
          00000000000000000000
          00000000000000000000
00000094  00000062 R                dd      offset flat:a0001
00000098  00000064 R                dd      offset flat:a0002
0000009C  00000068 R                dd      offset flat:a0003
000000A0  00000070 R                dd      offset flat:a0004
000000A4  00000080 R                dd      offset flat:a0005
000000A8  0001 0001 0001            dw      type a0001,length a0001,size a0001
000000AE  0002 0001 0002            dw      type a0002,length a0002,size a0002
000000B4  0004 0001 0004            dw      type a0003,length a0003,size a0003
000000BA  0008 0001 0008            dw      type a0004,length a0004,size a0004
000000C0  000A 0001 000A            dw      type a0005,length a0005,size a0005

000000C6  0000000B [        a0011   byte    11 dup(?),?
           00
          ] 00
000000D2  0000000C [        a0012   word    12 dup(?),?
           0000
          ] 0000
000000EC  0000000D [        a0013   dword   13 dup(?),?
           00000000
          ] 00000000
00000124  0000000E [        a0014   qword   14 dup(?),?
           0000000000000000
          ]
          0000000000000000
0000019C  0000000F [        a0015   tbyte   15 dup(?),?
           00000000000000000000
          ]
          00000000000000000000
0000023C  000000C6 R                dd      offset flat:a0011
00000240  000000D2 R                dd      offset flat:a0012
00000244  000000EC R                dd      offset flat:a0013
00000248  00000124 R                dd      offset flat:a0014
0000024C  0000019C R                dd      offset flat:a0015
00000250  0001 000B 000B            dw      type a0011,length a0011,size a0011
00000256  0002 000C 0018            dw      type a0012,length a0012,size a0012
0000025C  0004 000D 0034            dw      type a0013,length a0013,size a0013
00000262  0008 000E 0070            dw      type a0014,length a0014,size a0014
00000268  000A 000F 0096            dw      type a0015,length a0015,size a0015

0000026E  00000064 R                dd      offset flat:lab001
00000272  FF04 0001 FF04            dw      type lab001,length lab001,size lab001


00000064                    lab001:
00000064  8D 1D 000000EC R          lea     ebx,a0013
0000006A  BB 000000EC R             mov     ebx,offset flat:a0013

0000006F  BB 000000CB R             mov     ebx,offset flat:a0011+5
00000074  BB 000000CB R             mov     ebx,offset flat:(a0011+5)
00000079  BB 000000CB R             mov     ebx,(offset flat:a0011)+5

0000007E  B8 00000008               mov     eax,type a0014
00000083  B8 0000000E               mov     eax,length a0014
00000088  B8 00000070               mov     eax,size a0014
```

2番目以降のデータはOFFSET，TYPE，LENGTH，SIZEの対象とならない

OFFSET，TYPE，LENGTH，SIZEの演算子は，データ定義時の最初のデータの情報を返す

ラベルはOFFSET演算子のみ有効な値を返す

OFFSETに対する演算は加減算のみ有効

以上のことから，LEA命令とOFFSET演算子は同じようにも見えますが，LEA命令が実際のプログラム実行によりアドレスを得るのに対して，OFFSET演算子はアセンブラ時にアドレスを得るという大きな違いがあります．また，LEA命令はシンボルのアドレス以外にもレジスタを使った間接アドレッシングの実効アドレスも取得できるというメリットがあります．さらにOFFSET演算子は，シンボルのアドレスのみですが，データ定義の初期値として使用できるため，メモリ上にアドレスのテーブルなどを作っておく場合に役立ちます．

(3) **TYPE**演算子，**LENGTH**演算子，**SIZE**演算子

これらの演算子は単項演算子で，この演算子を使用すると，アドレス以外のデータが定義されたときの属性が取得できます(リスト4)．

TYPE演算子は，BYTEやWORDといった型のバイト数を返します．たとえばシンボルがBYTEで定義されていたら1，ワードで定義されていたら2といったぐあいです．

LENGTH演算子は，定義された最初のinitializerのデータの個数を返します．また，SIZE演算子は，定義された最初のinitializerのバイト数，つまり「TYPE演算子の結果×LENGTH演算子の結果」の値を返します．

たとえば，

```
v01  dw  5 dup(0), 2 dup(0)
```

で定義されたv01に対して，各演算子の結果は，

TYPE演算子の結果＝2
LENGTH演算子の結果＝5
SIZE演算子の結果＝10

となります．

● 外部モジュールとのリンク

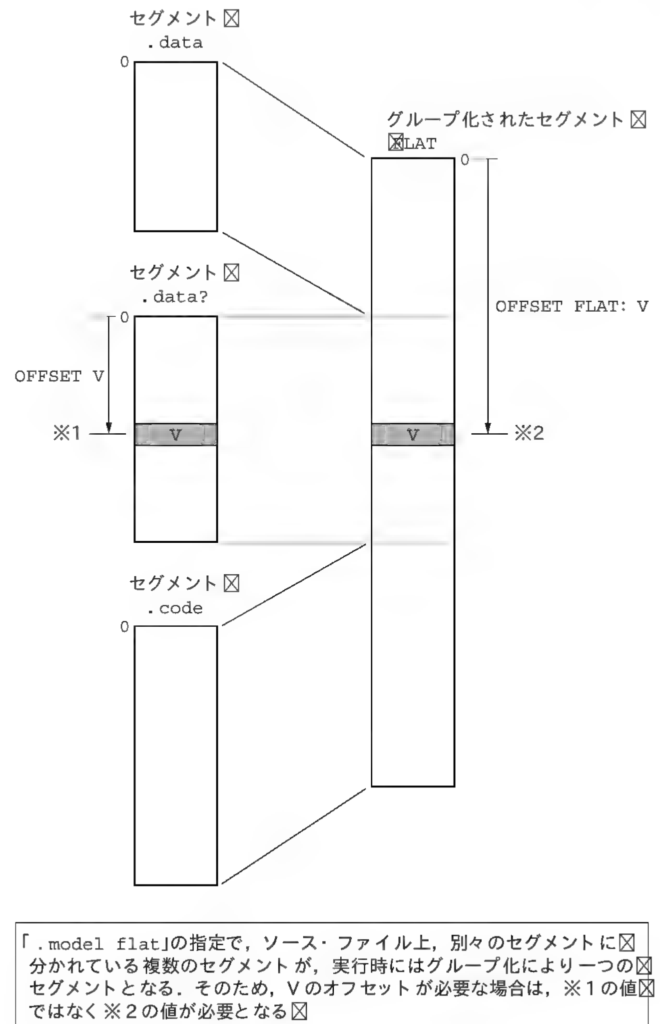
アセンブラのプログラムどうし，あるいはC(C++)のプログラムとリンクする場合，アセンブラのプログラムがほかのモジュール上にある変数をアクセスしたり，その逆にほかのモジュール上にあるプログラムがアセンブラの変数をアクセスする場合もあります．また，アセンブラのサブルーチンがほかのモジュールのプログラムから呼び出されることや，アセンブラのプログラムがほかのモジュールのサブルーチンを呼び出す場合もあります．

これらのモジュール間のシンボルのリンクは，リンカにより行われ，リンカにより外部モジュールを参照しているシンボルの最終的なアドレスが決定されます．

(1) ほかのモジュールへのシンボルの公開

PUBLICディレクティブを使い，ほかのモジュールへシンボルを公開します．PUBLICでは，すでに定義済みのシンボルをオペランドに記述するのみで，指定されたシンボルがほかのモジュールから使用可能になります．たとえば，バイト・データのval1とラベルのfunc1という二つのシンボルをほかのモジュールから使用可能にするためには，

```
PUBLIC  val1, func1
```



「.model flat」の指定で，ソース・ファイル上，別々のセグメントに分かれている複数のセグメントが，実行時にはグループ化により一つのセグメントとなる．そのため，Vのオフセットが必要な場合は，※1の値ではなく※2の値が必要となる

と記述します．

(2) ほかのモジュールのシンボルの参照

EXTERNディレクティブを使い，ほかのモジュールで定義されたシンボルを参照します．正確にはEXTERNディレクティブで参照の準備をします．EXTERNでは，ほかのモジュールで定義されたシンボルの名前とそのタイプ(型)を指定することで，指定されたシンボルが使用可能になります．たとえば，ほかのモジュールで定義されているバイト・データのval2とラベル(NEAR)のfunc2という二つのシンボルを，このモジュールで使用可能にするためには，

```
EXTERN  val2: byte, func2: near
```

と記述します．

(3) **PUBLIC**と**EXTERN**を使用する上での注意

以前述べた，アセンブル時のオプション/Cの指定には注意が必要です．C(C++)のようなシンボルの大文字と小文字を区別する言語とリンクする場合は，/Cxあるいは/Cpを指定する必要があります．

● サブルーチンとデータの構造の記述

MASMには，プログラムの構造を分かりやすく記述するための「プロシージャ」と「構造体，共用体」の定義機能をもっています．

(1) プロシージャの定義

アセンブラでサブルーチンを記述する場合，入り口をラベルで定義し，終わりをRET命令で表すような方法が使われます．しかし，この方法では，コメントをしっかり書かないと，サブルーチンの開始と終了がわからなくなる心配があります〔図5(a)〕．そのための機能として，MASMにはプロシージャの定義機能があります．プロシージャは，図5(b)に示すようサブルーチンの開始をPROCディレクティブ，サブルーチンの終了をENDPディレクティブで指定します．このPROCとENDPを使うことによって，サブルーチンの開始から終了までを明確にすることができます．これはデバック時や，後でプログラムを見るときなどに役立ちます．

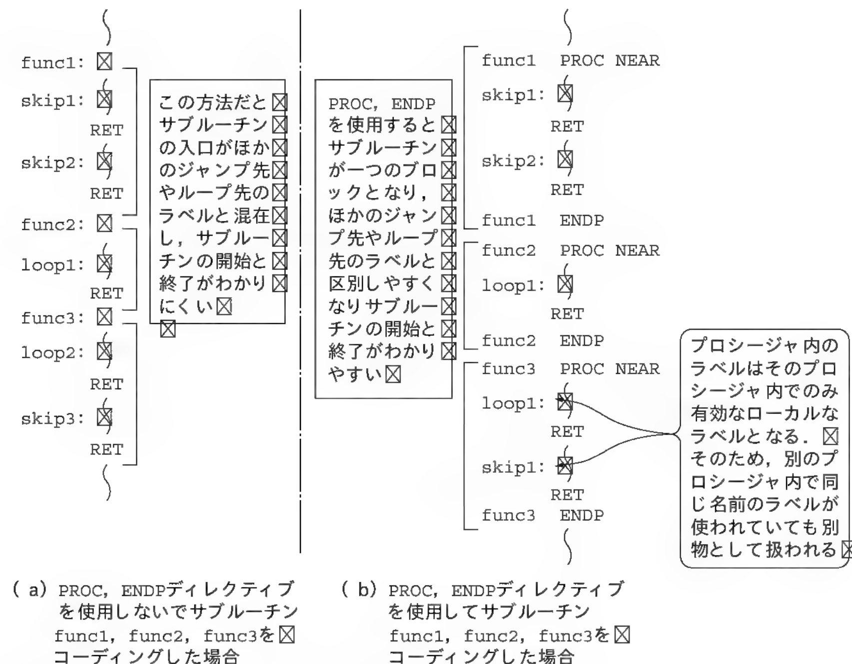
(2) 構造体，共用体の定義とアクセス

MASMでは，データとして構造体や共用体も使うことができます．これはC言語の構造体や共用体と同じ機能です．

構造体や共用体を使用する場合，まず型を宣言し，その型を使い変数を定義します．構造体と共用体の型の宣言は，構造体はSTRUCTディレクティブ，共用体はUNIONディレクティブで宣言の開始を指定します．そして各フィールドとなるデータを必要なだけ定義します．最後にENDSディレクティブで宣言を終わります．フィールドに名前を付ける場合，その名前は，同じ構造体あるいは共用体の中では一意である必要があります．構造体や共用体は，その中に別の構造体や共用体も定義できます．

構造体や共用体の変数を定義するには，通常のデータと同じように行います．ただし，型の部分が事前に宣言された構造体や共用体の型となり，初期値の指定は<>あるいは{ }で囲まれたカンマで区切られた複数個の値となります．

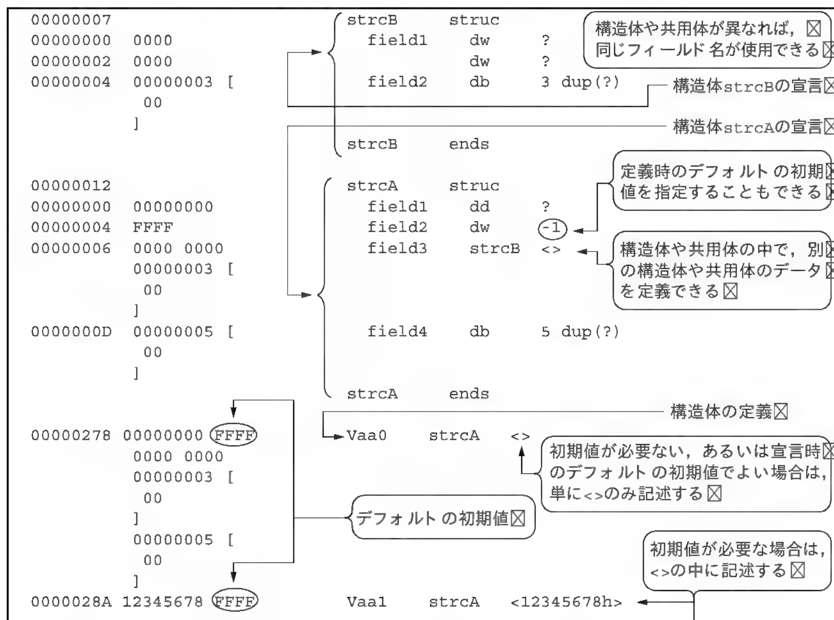
構造体や共用体上のフィールドのアクセスは，ピリオドの後にフィールド名を記述する方法で行われます．実際の構造体，共用体の使い方はリスト5を参考にしてください．



(a) PROC，ENDPディレクティブを使用しないでサブルーチンfunc1，func2，func3をコーディングした場合
(b) PROC，ENDPディレクティブを使用してサブルーチンfunc1，func2，func3をコーディングした場合

図5 **PROC**ディレクティブと**ENDP**ディレクティブ

リスト5 MASMの構造体と共用体



```
00000007                          strcB      struc
00000000  0000                      field1     dw      ?
00000002  0000                                 dw      ?
00000004  00000003 [                field2     db      3 dup(?)
           00
          ]
                                  strcB      ends

00000012                          strcA      struc
00000000  00000000                  field1     dd      ?
00000004  FFFF                      field2     dw      -1
00000006  0000 0000                 field3     strcB   <>
          00000003 [
           00
          ]
0000000D  00000005 [                field4     db      5 dup(?)
           00
          ]
                                  strcA      ends

00000278 00000000 FFFF            Vaa0     strcA     <>
          0000 0000
          00000003 [
           00
          ]
          00000005 [
           00
          ]
0000028A 12345678 FFFF            Vaa1     strcA     <12345678h>
```


おおぬき・ひろゆき　大貫ソフトウェア設計事務所

リスト 5
MASM の構造体と共用体
（つづき）

```
                 0000 0000
                 00000003 [
                  00
                 ]
                 00000005 [
                  00
                 ]
 0000029C 12345678 0500          Vaa2    strcA   <12345678h,500h>
                 0000 0000
                 00000003 [
                  00
                 ]
                 00000005 [
                  00
                 ]
 000002AE 12345678 0500          Vaa3    strcA   <12345678h,500h,,<1,2,3,4,5>>
                 0000 0000
                 00000003 [
                  00
                 ] 01 02 03 04
                 05
 000002C0 12345678 0500          Vaa4    strcA   <12345678h,500h,<6,7,<10,11,12>>,<11h,22h>>
                 0006 0007 0A
                 0B 0C 11 22
                 00000003 [
                  00
                 ]
 000002D2 12345678 0500          Vaa5    strcA   <12345678h,500h,<32,33,<34>>>
                 0020 0021 22
                 00000002 [
                  00
                 ]
                 00000005 [
                  00
                 ]

 00000007                        strcDT    struc
                                   struct DATE
 00000000  0000                      year      dw   ?
 00000002  00                        month     db   ?
 00000003  00                        day       db   ?
                                   ends
                                   struct TIME
 00000004  00                        hour      db   ?
 00000005  00                        minute    db   ?
 00000006  00                        second    db   ?
                                   ends
                                 strcDT    ends

 000002E4 0000 00 00 00          Vdt   strcDT   <>
          00 00

 00000004                        unonA     union
 00000000  00000000                field1    dd      ?
                                   field2    db      4 dup(?)
                                 unonA     ends

 000002EB 00000000               Vbb1    unonA   <>
 000002EF 12345678               Vbb2    unonA   <12345678h>


 0000008D  66| A1                        mov     ax,Vaa4.field2
           000002C4 R
 00000093  A0 000002CB R                 mov     al,Vaa4.field3.field2+1

 00000098  BB 000002C0 R                 mov     ebx,offset flat:Vaa4
 0000009D  8B 03                         mov     eax,(strcA ptr [ebx]).field1
 0000009F  8B 03                         mov     eax,[ebx].strcA.field1
 000000A1  8A 43 0C                      mov     al,(strcA ptr [ebx]).field3.field2+2
 000000A4  8A 43 0C                      mov     al,[ebx].strcA.field3.field2+2

 000000A7  A0 000002E7 R                 mov     al,Vdt.DATE.day
 000000AC  BE 000002E4 R                 mov     esi,offset flat:Vdt
 000000B1  8A 46 03                      mov     al,(strcDT ptr [esi]).DATE.day
 000000B4  8A 46 03                      mov     al,[esi].strcDT.DATE.day

 000000B7  C7 05 000002EF R              mov     vbb2.field1,87654321h
           87654321
 000000C1  A0 000002EF R                 mov     al,vbb2.field2
 000000C6  A0 000002F0 R                 mov     al,vbb2.field2+1
 000000CB  A0 000002F1 R                 mov     al,vbb2.field2+2
 000000D0  A0 000002F2 R                 mov     al,vbb2.field2+3
```



第16回
GCC2.95から追加変更のあったオプションの補足と検証(その4)

岸 哲夫

今回は本連載第14回(本誌2004年2月号に掲載)に続いて，GCC2.95から追加変更のあったオプションの補足と検証を行います．今回は特に「最適化オプション」について扱います．

(筆者)

● -falign-functions

関数の境界そろえを行います．`-falign-functions=32`と指定すると関数を32バイト境界に配置します．`-falign-functions`を指定するとマシンに依存するデフォルト値が設定されます．

PowerPCのG5では，ホット・ループ，分岐，または分岐ターゲットを32バイト境界にそろえることが推奨されています．

ソースと生成されたコードをリスト1～リスト4に示します．

32バイトで境界そろえを行った場合は，生成されたアセンブラ・ソースに“.p2align 5,,31”という命令文が入ります．これは「2の5乗境界でそろえるが，31バイト以上のNOP命令で埋めなければならないなら境界そろえを行わない」という意味です．

64バイトで境界そろえを行っている場合，“.p2align 6,,63”という命令文が定義されています．境界そろえを行わない場合には，その命令は定義されていません(リスト5)．

なお，`-falign-functions`だけ指定した場合，i686環境下では16ビット境界にそろえられます．この場合，“.p2align 4,,15”という命令文が定義されています．

次に示すように`-falign-xxxx`というオプションがいくつかありますが，基本はこれと同じです．

▶ 境界に合わせてループのアラインメントを保証する

`-falign-loops`

`-falign-loops=n`

▶ 境界に合わせてラベルを配置

`-falign-labels`

`-falign-labels=n`

▶ 境界に合わせてジャンプ命令を配置

`-falign-jumps`

`-falign-jumps=n`

リスト1 関数の境界揃えを行う例(test207.c)

```
#include <stdio.h>
/*
 *関数の境界揃え
 */
int test1();
int test2();
int test21();
int main(int argc, char* argv[])
{
    printf("%d¥n",test1());
    printf("%d¥n",test2());
    return    0;
}
int test1()
{
    return    100;
}
int test2()
{
    printf("%d¥n",test21());
    return    200;
}
int test21()
{
    return    300;
}
```

● -fbranch-probabilities

詳細は，まだ触れていないデバッグ・オプションの項で説明しますが，`-fprofile-arcs`というデバッグ用のオプションがあります．そのオプションを指定すると，実行時にプログラム・コードに存在する分岐のどちらを通るかを記録します．その後，`-fbranch-probabilities`オプションを付加してコンパイルを行うと，あまり通らない分岐と必ず通る分岐を認識して強力な最適化を行います(リスト6～リスト8)．

非常に単純な例で，適切なものではありませんが，次のような元ソースにある意味のないコードがアセンブラ・ソースからは排除されています．

```
dummy++;
dummy++;
dummy++;
dummy++;
dummy  =  0;
```

このようなソースではなく，もっと複雑な分岐を持つソースで，実行に応じた最適化を行うことで役立つと思います．実際にはデバッグ時に役立つオプションなので，そのときにくわしく解説します．

リスト 2　32バイト境界そろえにするオプションを付けて生成されたアセンブラ・ソース(test207a.s)

```
    .file     "test207.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
    .p2align 5,,31
.globl main
    .type     main, @function
main:
    pushl       %ebp
    movl %esp, %ebp
    subl $8,    %esp
    andl $-16, %esp
    movl $0,    %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0,     %eax
    leave
    ret
    .size     main, .-main
    .p2align 5,,31
.globl test1
    .type     test1, @function
test1:
    pushl        %ebp
    movl %esp,   %ebp
    movl $100,   %eax
    popl %ebp
    ret
    .size     test1, .-test1
    .p2align 5,,31
.globl test2
    .type     test2, @function
test2:
    pushl        %ebp
    movl %esp,   %ebp
    subl $8,     %esp
    call test21
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $200,  %eax
    leave
    ret
    .size     test2, .-test2
    .p2align 5,,31
.globl test21
    .type     test21, @function
test21:
    pushl       %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test21, .-test21
    .ident    "GCC: (GNU) 3.3"
```

リスト 3　64バイト境界そろえにするオプションを付けて生成されたアセンブラ・ソース(test207b.s)

```
    .file     "test207.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
    .p2align 6,,63
.globl main
    .type     main, @function
main:
    pushl       %ebp
    movl %esp, %ebp
    subl $8,    %esp
    andl $-16, %esp
    movl $0,    %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0,     %eax
    leave
    ret
    .size     main, .-main
    .p2align 6,,63
.globl test1
    .type     test1, @function
test1:
    pushl        %ebp
    movl %esp,   %ebp
    movl $100,   %eax
    popl %ebp
    ret
    .size     test1, .-test1
    .p2align 6,,63
.globl test2
    .type     test2, @function
test2:
    pushl        %ebp
    movl %esp,   %ebp
    subl $8,     %esp
    call test21
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $200,  %eax
    leave
    ret
    .size     test2, .-test2
    .p2align 6,,63
.globl test21
    .type     test21, @function
test21:
    pushl       %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test21, .-test21
    .ident    "GCC: (GNU) 3.3"
```

リスト 4　オプションなしで生成されたアセンブラ・ソース(test207c.s)

```
    .file     "test207.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl       %ebp
    movl %esp, %ebp
    subl $8,    %esp
    andl $-16, %esp
    movl $0,    %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0,     %eax
    leave
    ret
    .size     main, .-main
.globl test1
    .type     test1, @function
test1:
    pushl        %ebp
    movl %esp,   %ebp
    movl $100,   %eax
    popl %ebp
    ret
    .size     test1, .-test1
.globl test2
    .type     test2, @function
test2:
    pushl        %ebp
    movl %esp,   %ebp
    subl $8,     %esp
    call test21
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $200,  %eax
    leave
    ret
    .size     test2, .-test2
.globl test21
    .type     test21, @function
test21:
    pushl       %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test21, .-test21
    .ident    "GCC: (GNU) 3.3"
```

リスト 5　境界そろえ値を省略したオプションを付けて生成されたアセンブラ・ソース(test207.s)

```
    .file     "test207.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
    .p2align 4,,15
.globl main
    .type     main, @function
main:
    pushl       %ebp
    movl %esp, %ebp
    subl $8,    %esp
    andl $-16, %esp
    movl $0,    %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0,     %eax
    leave
    ret
    .size     main, .-main
    .p2align 4,,15
.globl test1
    .type     test1, @function
test1:
    pushl        %ebp
    movl %esp,   %ebp
    movl $100,   %eax
    popl %ebp
    ret
    .size     test1, .-test1
    .p2align 4,,15
.globl test2
    .type     test2, @function
test2:
    pushl        %ebp
    movl %esp,   %ebp
    subl $8,     %esp
    call test21
    movl %eax,  4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $200,  %eax
    leave
    ret
    .size     test2, .-test2
    .p2align 4,,15
.globl test21
    .type     test21, @function
test21:
    pushl       %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test21, .-test21
    .ident    "GCC: (GNU) 3.3"
```

リスト 6　分岐が起こる経路を推測して行われる最適化例(test208.c)

```
/*
* 分岐が起こる経路を推測して行われる最適化
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{
    int   Wday;
    int   Wday_save;
    int  dummy;
    enum Days
    {
        Sunday     =    0,
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
        Friday,
        Saturday
    };
    Wday =    1;
    switch    (Wday)
    {
        case Sunday:
            Wday_save =    Sunday;
            printf("%s¥n","Sunday");
            dummy++;
            dummy++;
            dummy++;
            dummy++;
            dummy     =    0;
            break;
        case Monday:
            Wday_save =    Monday;
            printf("%s¥n","Monday");
            break;
        case Tuesday:
            Wday_save =    Tuesday;
            printf("%s¥n","Tuesday");
            break;
        case Wednesday:
            Wday_save =    Wednesday;
            printf("%s¥n","Wednesday");
            break;
        case Thursday:
            Wday_save =    Thursday;
            printf("%s¥n","Thursday");
            break;
        case Saturday:
            Wday_save =    Saturday;
            printf("%s¥n","Saturday");
            dummy++;
            dummy++;
            dummy++;
            dummy++;
            dummy     =    0;
            break;
        default:
            Wday_save =    Saturday;
    }
    printf("%d¥n",Wday_save);
    return    0;
}
```

リスト 7　-fprofile-arcs を付加して生成されたアセンブラ・ソース(test208a.s)

```
    .file     "test208.c"
    .section  .rodata
.LC0:
    .string   "Sunday"
.LC1:
    .string   "%s¥n"
.LC2:
    .string   "Monday"
.LC3:
    .string   "Tuesday"
.LC4:
    .string   "Wednesday"
.LC5:
    .string   "Thursday"
.LC6:
    .string   "Saturday"
.LC7:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl            %ebp
    movl %esp,       %ebp
    subl $24,        %esp
    andl $-16,       %esp
    movl $0,         %eax
    subl %eax,       %esp
    movl $5,         -4(%ebp)
    cmpl $6,         -4(%ebp)
    ja   .L9
    movl -4(%ebp),   %eax
    sall $2,    %eax
    movl .L10(%eax), %eax
    jmp  *%eax
    .section  .rodata
    .align 4
    .align 4
.L10:
    .long     .L3
    .long     .L4
    .long     .L5
    .long     .L6
    .long     .L7
    .long     .L9
    .long     .L8
    .text
.L3:
    movl $0,   -8(%ebp)
    movl $.LC0, 4(%esp)
    movl $.LC1,  (%esp)
    call printf
    jmp  .L2
.L4:
    movl $1,   -8(%ebp)
    movl $.LC2, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    jmp  .L2
.L5:
    movl $2,   -8(%ebp)
    movl $.LC3, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    jmp  .L2
.L6:
    movl $3,   -8(%ebp)
    movl $.LC4, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    jmp  .L2
.L7:
    movl $4,   -8(%ebp)
    movl $.LC5, 4(%esp)
    movl $.LC1,  (%esp)
    call printf
    jmp  .L2
.L8:
    movl $6,   -8(%ebp)
    movl $.LC6, 4(%esp)
    movl $.LC1,  (%esp)
    call printf
    jmp  .L2
.L9:
    movl $6,  -8(%ebp)
.L2:
    movl -8(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC7, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

リスト 8　-fbranch-probabilities を付加して生成されたアセンブラ・ソース(test208b.s)

```
    .file     "test208.c"
    .section  .rodata
.LC0:
    .string   "Sunday"
.LC1:
    .string   "%s¥n"
.LC2:
    .string   "Monday"
.LC3:
    .string   "Tuesday"
.LC4:
    .string   "Wednesday"
.LC5:
    .string   "Thursday"
.LC6:
    .string   "Saturday"
.LC7:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl      %ebp
    movl %esp, %ebp
    subl $24,  %esp
    andl $-16, %esp
    movl $0,   %eax
    subl %eax, %esp
    movl $1,   -4(%ebp)
    cmpl $6, -4(%ebp)
    ja   .L9
    movl -4(%ebp), %eax
    sall $2, %eax
    movl .L10(%eax), %eax
    jmp  *%eax
    .section  .rodata
    .align 4
    .align 4
.L10:
    .long     .L3
    .long     .L4
    .long     .L5
    .long     .L6
    .long     .L7
    .long     .L9
    .long     .L8
    .text
.L3:
    movl $0,   -8(%ebp)
    movl $.LC0, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    leal -12(%ebp), %eax
    incl (%eax)
    leal -12(%ebp), %eax
    incl (%eax)
    leal -12(%ebp), %eax
    incl (%eax)
    leal -12(%ebp), %eax
    incl (%eax)
    movl $0, -12(%ebp)
    jmp  .L2
.L4:
    movl $1,   -8(%ebp)
    movl $.LC2, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    jmp  .L2
.L5:
    movl $2,   -8(%ebp)
    movl $.LC3, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    jmp  .L2
.L6:
    movl $3,   -8(%ebp)
    movl $.LC4, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    jmp  .L2
.L7:
    movl $4,   -8(%ebp)
    movl $.LC5, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    jmp  .L2
.L8:
    movl $6,   -8(%ebp)
    movl $.LC6, 4(%esp)
    movl $.LC1, (%esp)
    call printf
    leal -12(%ebp), %eax
    incl (%eax)
    leal -12(%ebp), %eax
    incl (%eax)
    leal -12(%ebp), %eax
    incl (%eax)
    leal -12(%ebp), %eax
    incl (%eax)
    movl $0, -12(%ebp)
    jmp  .L2
.L9:
    movl $6, -8(%ebp)
.L2:
    movl -8(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC7, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

リスト 9
不要なグローバル・データを bss 領域に配置する例(test209.c)

```
//不要なグローバルデータをｂｓｓ領域に配置
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int test1();
int test2();
int test3();
long     testdata;
int main(int argc, char* argv[])
{
    printf("%d¥n",test1());
    printf("%d¥n",test2());
    return    0;
}
int test1()
{
    return    100;
}
int test2()
{
    return    200;
}
int test3()
{
    return    300;
}
```

リスト 10
オプションを付加しないで生成したアセンブラ・ソース(test209a.s)

```
    .file     "test209.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl      %ebp
    movl %esp, %ebp
    subl $8,   %esp
    andl $-16, %esp
    movl $0,   %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
.globl test1
    .type     test1, @function
test1:
    pushl      %ebp
    movl %esp, %ebp
    movl $100, %eax
    popl %ebp
    ret
    .size     test1, .-test1
.globl test2
    .type     test2, @function
test2:
    pushl      %ebp
    movl %esp, %ebp
    movl $200, %eax
    popl %ebp
    ret
    .size     test2, .-test2
.globl test3
    .type     test3, @function
test3:
    pushl      %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test3, .-test3
    .comm     testdata,4,4
    .ident    "GCC: (GNU) 3.3"
```

● -fdata-sections

このオプションを付加すると，使われていない不要なグローバル・データを，共通領域ではなく .bss 領域に配置します．

出力されたアセンブラからもわかるように，オプションを付加しない場合には .comm 領域に配置され，付加した場合には .bss 領域に配置されます(リスト 9～リスト 11)．

● -ffunction-sections

このオプションを付加すると，使われていない関数のマッピング情報を明示的に除去します．

オプションを付加した場合には，関数の配置情報が除去されています(リスト 12～リスト 14)．

次のような疑似命令が使用している関数 test1，test2 に追加されていて，test3 には追加されていません．

```
.section  .text.test1,"ax",@progbits
```

これは,「 割り当て可能で実行可能な test1 を .text に配置する」という意味です．

リスト 11
オプションを付加して生成したアセンブラ・ソース（test209b.s）

```
    .file     "test209.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $8,   %esp
    andl $-16, %esp
    movl $0,   %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0,   %eax
    leave
    ret
    .size     main, .-main
.globl test1
    .type     test1, @function
test1:
    pushl     %ebp
    movl %esp, %ebp
    movl $100, %eax
    popl %ebp
    ret
    .size     test1, .-test1
.globl test2
    .type     test2, @function
test2:
    pushl     %ebp
    movl %esp, %ebp
    movl $200, %eax
    popl %ebp
    ret
    .size     test2, .-test2
.globl test3
    .type     test3, @function
test3:
    pushl     %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test3, .-test3
.globl testdata
    .section
.bss.testdata,"aw",@nobits
    .align 4
    .type     testdata, @object
    .size     testdata, 4
testdata:
    .zero     4
    .ident    "GCC: (GNU) 3.3"
```

● -fdelayed-branch

RISCチップではCPU内部の最適化のために「遅延スロット」というしくみを持っているものがあります．簡単にいえば分岐命令の後の命令をつねに実行することで，効率を高める方式です．

CPUが遅延スロットに対応しているならば，このオプションを付加してコンパイルすることで，RISCチップの機能を有効に使うことができます．ただし，Zaurus（ザウルス）などに使われているARMアーキテクチャのCPUには「遅延スロット」がありません．

● -fdelete-null-pointer-checks

このオプションを付けると，グローバル・データ・フロー分

リスト 12　不要な関数をマッピングしない例（test210.c）

```
//不要な関数をマッピングしない
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int test1();
int test2();
int test3();
int main(int argc, char* argv[])
{
    printf("%d¥n",test1());
    printf("%d¥n",test2());
    return    0;
}
int test1()
{
    return    100;
}
int test2()
{
    return    200;
}
int test3()
{
    return    300;
}
```

リスト 13　オプションを付加しないで生成したアセンブラ・ソース（test210a.s）

```
    .file     "test210.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $8,   %esp
    andl $-16, %esp
    movl $0,   %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
.globl test1
    .type     test1, @function
test1:
    pushl     %ebp
    movl %esp, %ebp
    movl $100, %eax
    popl %ebp
    ret
    .size     test1, .-test1
.globl test2
    .type     test2, @function
test2:
    pushl     %ebp
    movl %esp, %ebp
    movl $200, %eax
    popl %ebp
    ret
    .size     test2, .-test2
.globl test3
    .type     test3, @function
test3:
    pushl     %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test3, .-test3
    .ident    "GCC: (GNU) 3.3"
```

リスト 14　オプションを付加して生成したアセンブラ・ソース（test210b.s）

```
    .file     "test210.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .section  .text.main,"ax",@progbits
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $8,   %esp
    andl $-16, %esp
    movl $0,   %eax
    subl %eax, %esp
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .section  .text.test1,"ax",@progbits
.globl test1
    .type     test1, @function
test1:
    pushl     %ebp
    movl %esp, %ebp
    movl $100, %eax
    popl %ebp
    ret
    .size     test1, .-test1
    .section  .text.test2,"ax",@progbits
.globl test2
    .type     test2, @function
test2:
    pushl     %ebp
    movl %esp, %ebp
    movl $200, %eax
    popl %ebp
    ret
    .size     test2, .-test2
    .section  .text.test3,"ax",@progbits
.globl test3
    .type     test3, @function
test3:
    pushl     %ebp
    movl %esp, %ebp
    movl $300, %eax
    popl %ebp
    ret
    .size     test3, .-test3
    .ident    "GCC: (GNU) 3.3"
```

析を使用して，不要なヌル・ポインタ・チェックを特定し，これを削除します．最適化オプションの-O2，-O3，-Osを付けてコンパイルすれば，このオプションがデフォルトで有効になります(リスト15〜リスト20)．

リスト16　-Osオプションを付加して生成したアセンブラ・ソース(test211_Os.s)

```
    .file     "test211.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC1:
    .string   "p is NULL-PO"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    pushl     $.LC1
    call      puts
    popl      %eax
    xorl      %eax, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

リスト17　-O1オプションを付加して生成したアセンブラ・ソース(test211_O1.s)

```
    .file     "test211.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC1:
    .string   "p is NULL-PO"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $.LC1, (%esp)
    call      puts
    movl      $0, %eax
    movl      %ebp, %esp
    popl      %ebp
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

リスト19　-O3オプションを付加して生成したアセンブラ・ソース(test211_O3.s)

```
    .file     "test211.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC1:
    .string   "p is NULL-PO"
    .text
    .p2align  4,,15
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $.LC1, (%esp)
    call      puts
    movl      %ebp, %esp
    xorl      %eax, %eax
    popl      %ebp
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

リスト15　不要なヌル・ポインタ・チェックを削除する例(test211.c)

```
//不要なヌル・ポインタ・チェックを削除する例
#include <stdio.h>
int main()
{
    int *p = (int *) NULL;
    if (p)
    {
        printf ("p is not NULL¥n");
    }
    else
    {
        printf ("p is NULL-PO¥n");
    }
    return 0;
}
```

リスト18　-O2オプションを付加して生成したアセンブラ・ソース(test211_O2.s)

```
    .file     "test211.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC1:
    .string   "p is NULL-PO"
    .text
    .p2align  4,,15
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $.LC1, (%esp)
    call      puts
    movl      %ebp, %esp
    xorl      %eax, %eax
    popl      %ebp
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

リスト20　-O0オプションを付加して生成したアセンブラ・ソース(test211_O0.s)

```
    .file     "test211.c"
    .section  .rodata
.LC0:
    .string   "p is not NULL¥n"
.LC1:
    .string   "p is NULL-PO¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $0, -4(%ebp)
    cmpl      $0, -4(%ebp)
    je   .L2
    movl      $.LC0, (%esp)
    call      printf
    jmp  .L3
.L2:
    movl      $.LC1, (%esp)
    call      printf
.L3:
    movl      $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

リスト 21　浮動小数点演算の精度が変わってしまうような最適化を行う例(test212.c)

```
//浮動小数点演算の精度が変わってしまうような最適化をする例
#include <stdio.h>
#include <math.h>
const float f1 = 3.1212312312312312f;
const float f2 = 6.5432165432165432f;

float func(float a)
{
    return a * f1 / f2;
}
```

リスト 23　最適化して生成したアセンブラ・ソース(test211a.s)

```
.   .file     "test212.c"
.globl f1
    .section  .rodata
    .align 4
    .type     f1, @object
    .size     f1, 4
f1:
    .long     1078444609
.globl f2
    .align 4
    .type     f2, @object
    .size     f2, 4
f2:
    .long     1087463944
    .text
.globl func
    .type     func, @function
func:
    pushl     %ebp
    movl      %esp, %ebp
    flds      8(%ebp)
    fmuls     f1
    fdivs     f2
    popl      %ebp
    ret
    .size     func, .-func
    .ident    "GCC: (GNU) 3.3"
```

最適化なしのオプション-O0以外は，

```
printf ("p is not NULL¥n");
```

が不要なコードとして削除されています．

● -fexpensive-optimizations

このオプションを付けると，最適化によってコンパイル時間をかけることに見合わないような細かい最適化を行います．

しかし，塵も積もれば…というわけで，たとえばWebブラウザのMozillaなどでは，このオプションを付けることで微妙に速くなるようです．

● -ffast-math

このオプションを指定すると，次のオプションを一度に指定したことになります．

```
-fno-math-errno
-funsafe-math-optimizations
-fno-trapping-math
-ffinite-math-only
-fno-signaling-nans
```

結果として浮動小数点演算の精度が変わってしまうような最適化でも行うようになります．プログラムの用途によっては精度を変えてしまうと困るかもしれません．そこで意図的に，このオプションで指定するわけです(リスト 21～リスト 23)．

リスト 21にある式中の"f1 / f2"は定数にできます．そこで計算式を変更しています．ただ，この方法では当然，精度も変わってしまいます．

なお，最適化に際しては次のオプションを同時に付加しました．

```
# gcc test212.c -S -ffast-math
                    -fomit-frame-pointer -O3
```

＊　　　＊

次回も，引き続き「最適化オプション」の補足を行います．

リスト 22　生成したアセンブラ・ソース(test211.s)

```
    .file     "test212.c"
.globl f1
    .section  .rodata
    .align 4
    .type     f1, @object
    .size     f1, 4
f1:
    .long     1078444609
.globl f2
    .align 4
    .type     f2, @object
    .size     f2, 4
f2:
    .long     1087463944
    .section  .rodata.cst4,"aM",@progbits,4
    .align 4
.LC96:
    .long     1056193456
    .text
    .p2align  4,,15
.globl func
    .type     func, @function
func:
    flds .LC96
    fmuls     4(%esp)
    ret
    .size     func, .-func
    .ident    "GCC: (GNU) 3.3"
```

きし・てつお

IPパケットの隙間から

バージョン・アップしなかったことによる苦労

祐安 重夫

仕事をしていたら，突然CRTの画面が別のものに切り替わってしまった．CRTには入力が二つあり，そのとき使用していたものとは別のPCの画面に切り替わったのだ．それまで仕事をしていたPCはどうなったかというと，そのまま死んでしまったようで，電源は入るがbootしなくなってしまった．

このPCは日常の作業用マシンであり，さらに三つあるドメインのうちの一つのDNS，メール・サーバ，WWWサーバでもあったのでたいへんである．大事なファイルは読めなくなるし，急ぎの仕事を一時的にストップせざるを得なくなった．

ところでこのマシンは，この連載の2002年4月号でも書いたのだが，一度同じ症状になったという前科をもっている．そのときはマザーボードの故障と判断して，UNIX系に強いショップで組み立て済みのキットを見つけ，それを購入してハードディスクを入れ換えたら，見事に動作してしまった．

今回もマザーボードの故障であることはまちがいないようだったので，秋葉原まで出かけて新しいマシンを１台購入し，同じ手段で対応しようとしたが，今度はそう簡単にはいかなかった．何しろ故障したマシンのOSは，Red Hat 6.2という骨董品的な代物であり，さすがにもう最近のハードウェアには対応しなくなりつつあったのだ．

とりあえず，価格的には2年前の2/3程度で，仕様的にはかなり高機能のものが購入できたが，内蔵の40Gバイトのハードディスクを外して，13Gバイトのもの2台に変更するのはちょっともったいない．それでも交換して，とりあえず動作させてみることにした．

1度目のbootは何とかできて，ハードディスクの中身は無事であることが確認できた．

ところが2度目にbootしようとしたら，ハードディスクが`lost interrupt`というエラーを連発し始めた．これはWebで検索した結果，LILOに対して`linux noapic`と入力すればだいじょうぶだということがわかった．後は`/etc/lilo.conf`を書き換えてやれば，次からは自動的に処理される．ここまでで，かなり時間を費やしてしまった．

やっとbootできたと思ったら，オンボードのネットワーク・インターフェースが新しすぎて認識されないようだ．しかたがないので故障したほうのマシンからネットワーク・インターフェース・カードを抜いて（これはオンボードではなかった）挿してみたら，どうにか認識された．

オンボードのネットワーク・インターフェースは新しいマシンでは新規デバイスとして認識されていたのだが，実際には初期化がうまくいかずタイム・アウトになってしまい，動作しなかったのだ．今どき，ネットワークが動かないことには，実際には何の役にも立たないのと同様である．ましてやこのマシンはDNS，メール・サーバ，WWWサーバとして動作させようとしていたのだから，なおさらである．

さらに大事なファイルをほかのマシンにバックアップしておかないと，これからの作業にも差し障る．ネットワークが動作し始めたので，さっそくデータのバックアップを行ったのはいうまでもない．こうしてネットワークが動作するまでに，またかなりの時間をむだにしてしまった．

さて，ここまでくれば後に残るのはXの設定である．しかし，予想どおり，オンボードのビデオ・インターフェースは，新しすぎて`Xconfigurator`や`XFSetup`では設定対象に入っていない．とりあえずSVGAにしてみても，1600×1200では動作してくれないようだ．

ここでも故障したマシンのビデオ・インターフェースを持ってきて入れてやれば，動作するのではないかと思うのは当然である．そこで外して挿してみようとして，気がついた．なんとPCIではないのだ．故障したマシンにはISAのインターフェースが一つだけ付いていて，そこに入っていたのだった．今どき，ISAバスを持ったPCなど，そう簡単に入手できるわけがない．

さらに原因不明なのだが，Xの設定を試みた後，突然コンソールがlogin画面になったとたんに点滅するようになってしまったが，いつのまにか直ってしまった．こうなったら新しい（かつ古い仕様の）ビデオ・カードを買い，差し換えてみるしかないが，とりあえず，すでに深夜だし，仕事はたまっているし，この原稿も締め切りをとうに過ぎているという問題もある．これは後日にまわすことにした．

ビデオ・カードはあらためて買いにいくとして，今はあまり使い慣れていないRed Hat 9の上で仕事を進めていくしかないようだ．実際に，この原稿もいつもとは違った使い慣れない環境で書いているので，かなり時間がかかってしまった．

しかし，本質的な問題は，Red Hat 6.2などという骨董品的な環境を，使い慣れているからという理由でそのままにしていたことにあるのかもしれない．今やThe Fedora Legacy Projectでさえ，サポートしていないOSである．

ちょうど，ずっと前にバージョン・アップしようとして買って，そのままにしてあるRed Hat 7.2がある．これならかろうじてThe Fedora Legacy Projectのサポート対象である．とりあえずこのあたりからバージョン・アップを始め，少しずつ移行していくしかないのかもしれない．

すけやす・しげお　インターメディアアクセス

シニアエンジニアの技術草子 四拾之段

◆近頃都に流行るもの

旭 征佑

● セルフ・スタンド

日本で初めてセルフ・スタンドが認められたのは，1998年4月の規制緩和によるものだった．当時のアメリカではすでにセルフ・スタンドが一般的だったが，日本で受け入れられるには時間がかかるとする向きが強かった．なにせ，運転席に座ったまま，「満タン」というだけで窓拭きから車内のごみ捨てまでしてくれるサービスが一般化していたからだ．しかし，5～6年たった現在，セルフ・スタンドの数はいつの間にか増えてきている．給油量が少なくても店員に気兼ねする必要がない，オイル交換を勧められることもなく煩わしくなくて良い，そして何より安いのが一番良いということだろう．

筆者も近くのセルフ・スタンドによく行く．鼻歌交じりに給油ノズルを握っていても，忙しく回っているメータに目を凝らして，1/100リットル単位できちっと給油を止めたりする．そんなこんなで手持ち無沙汰だった給油の待ち時間も少しだけ楽しい．タバコは吸わないので，吸殻を捨ててもらう必要はないし，ウインドウを拭いてもらえないのも，安いのだからまあ良しとしよう．残念なのは，明るく活気のある声が周りからまったく消えてしまったことぐらいだろうか．

そんなある日，家の近くのセルフでないスタンドでガソリンを入れることになった．そこではガソリンの値段はセルフと変わらなかったが，ウインドウを拭いてもらったうえ，ティッシュを一箱もらえた．何だか少し得をしたような気持ちになった．そういえば，以前はスタンプを押してもらい，ティッシュだ，洗剤だ，お米だなどと，いろいろもらっていた記憶がある．よく考えると，セルフ・スタンドを使うことで実は損をしているのではないだろうか．急にそう思えてきてしまった．

● セルフ・スタンドの事情

車に給油するノズルをもった一つ一つのスタンドをポンプと呼ぶらしい．日本のセルフ・スタンドのポンプにはいろいろな安全装置が付いていて，たいへん割高なのだという．給油ポンプにバックした車が激突し，ガソリンが噴出して火災になるのをアメリカ映画で見たことがある．人口密集地にスタンドが多い日本で，まさかそんな大きな事故を起こすわけにはいかない．誘導する店員がその場にいないのだから，万が一の車両衝突時でもポンプからガソリンが噴出しないための特別なくふうが必要だ．ほかにもいろいろな安全装置として，立ち上り遮断弁，給油起動制御，脱落防止，満量停止，可燃性蒸気回収，油種違い防止，給油ホース緊急離脱…など，各種の機能が必要だという．

車の給油キャップを開けたことがある人は，最初はちょっと驚くかもしれない．タンク内で気化したガソリンが，シュッと吹き出してくるからだ．静電気をためた体で触れば，スパークして引火する可能性もある．実際，この手の火傷の例は数多く報告されているらしい．そこで，セルフ・スタンドでは，車が入ってくるのをカメラで監視し，顧客の火気使用状況，静電気を逃がしたかどうかなどを確認し，危険性がないと判断して初めて給油バルブを開け，そして給油状況を監視し，給油が終わるとバルブを閉めるそうだ．こんなめんどうな作業を毎回すべて手作業の遠隔操作でこなしているのである．そのため，各給油ポンプには監視モニタ，インターホン，消火設備などの各種リモート・コントロール装置が付いている．コントロール・ブースと呼ばれる管理棟が別にあり，その中では数人の監視員が交代で監視カメラの映像を凝視し，コントロール・パネルを操作しているのだ．

一人で管理できるポンプは3～4台が限界だ．支払いや苦情などを受け付ける窓口の店員も必要だ．交代要員も含めてざっと計算すると，必要な人件費は今までとほとんど変わらなくなってしまう．一方，通常のスタンドを作るのに1億円かかるといわれるが，この種の設備を追加するとさらに5千万円もかかるという．

経費が増えるにもかかわらず，オイルや洗車，タイヤやバッテリ交換などの対面販売の売り上げがなくなってしまうわけだから，当然，経営は苦しくなるはずだ．結局は，コーヒーショップ，コンビニなどを併設して，売り上げ減をカバーするのに必死だという．これではガソリンが安くなるはずはない．

ある調査によれば，なぜセルフ・スタンドを使用するのかをリピータに聞いたところ，ガソリンが安いからと答えた人が70%を超えたという．しかし，現実には決して安いわけではない．リピータが安いと感じていたのは，以前のマスコミ報道や，セルフは安いという固定観念から受けた幻想だったに違いない．

● ネット・ショップ

最初はパソコン部品の販売が多かったネット・ショップも，最近では事務用品，本，そしてDVDなども一般化してきた．



女性では飲食・服飾関係もけっこう伸びてきているようだ．

このネット・ショップもセルフ・スタンドに似ているといったら，いいすぎだろうか．ネット・ショップも，NTT独占を打破した規制緩和で，定額のインターネットが普及し始めた1990年代後半に本格化した．当時，アメリカで成功の兆しを見せていたにもかかわらず，日本では抵抗が強くて成功しないだろうともいわれたが，数年たった現在は，リピータ層に支えられて着々と成功しつつある．都市部の若いユーザを中心に受け入れられている点，一般ユーザには予想できない多大な開発費用という経費が裏ではかかっているという点など，セルフ・スタンドと似ている点は多い．

ネットで物を買うと便利なのは事実だ．インターネットで買えば，いろんな種類の製品を比較できる．夜中にふと思い出したときでも，購入の手続きが取れる．翌日には配達され，重たいコピー用紙でもすぐ持ってきてくれる．上手に購入すれば送料も無料だ．こんな利点もあるため，リピータは多い．成功しているネット・ショップのリピート率が60%を超える場合も珍しくないようだ．

では，ネット・ショップは，はたして本当に安いのだろうか．日本特有の流通という中間コストが省かれ，見本品を置く必要がないし，在庫も最低限で良い．うるさいお客にからまれる機会も少ないだろうし，サイト管理もそんなにたいへんではなさそうだ．そう考えると，安くても良いような気がする．

実際，再販制度に守られている本やDVDなどは，うまくやれば数%程度の値引きになることもある．安くて便利ならば，それにこしたことはない．

うがった見かたをすると，リピータが多いというのは，逆に参加障壁が高いということなのかもしれない．申し込み手続きのわかりにくさ，IDやパスワード管理のめんどうくささ，各社で統一性のない注文手順など，新しいサイトに登録するたび，これらには眉をひそめる．これだけめんどうだと，ほかの競合ショップに登録する気がしなくなるからリピート率が高くなるともいえないだろうか．今のネット・ショップは，こういう使いにくさをがまんしてくれるユーザによってのみ支えられているのかもしれない．これは重要なことだ．なぜなら，がまんしてくれるユーザはごく一握りにすぎないからだ．



それにもかかわらず，ネット・ショップは成功すると，さらに投資して売り上げの向上を目指していく傾向がある．怖いのはネット・ショップの売り上げが無限に増えそうに見えることだ．総務省の「ITの経済分析に関する調査」の「電子商取引市場（B2C企業一個人）の推移」のデータによっても，毎年100%近い成長を続けており，今後もこの成長は続くという．たしかに投資しない手はない．

リピート・ユーザは煩雑な問題をがまんして多くの注文をする一方で，ネット・ショップは得た利益をユーザに還元することなく，どんどん投資して拡張を続けていく傾向がある．

安いと思ってリピータとして注文しているのなら，実はめんどうな思いをしたうえに高いものを買わされているのかもしれない．消費者として賢い目をもつことがとても重要になってくるはずだ．新しいしくみが一般庶民の期待にそぐわないのは，今も昔も変わらいない真実なのかもしれない．

あさひ・しょうすけ　テクニカル・ライター
イラスト　森 祐子

Engineering Life in

シリコンバレーでの人脈作り

最近，日本でもよく聞くことばに「ネットワーキング」がある．LANやコンピュータをつなぐネットワークではなく，人間関係を結ぶほうのネットワーキングだ．以前にも，さまざまな形でシリコンバレーで行われているネットワーキングについて紹介してきた．今回は，もっと具体的にどのような形でネットワーキング会議が行われたり，主催されているかを考えてみたい．

☆ 自己主張から始まる積極性

ネットワーキングは簡単に説明すると人脈を作ることだと思う．アメリカの人脈は横のつながりが主になるので，同じ立場や目線でお互いにギブ・アンド・テイクのやりとりがあり，何かしてもらったらいつかお返しをする…という発想がどこかにある．実際のネットワーキングは人と出会い，自己紹介をしたり，自分の興味をもっている分野や課題，そして得意としていることを手短に説明することから始まり，それが情報交換へとつながっていく．

実際にアメリカの展示会や会議に参加すると自己紹介などでハイペースに話を進める．まったく知り合いのいない場所で見知らぬ人に声をかけて名刺交換や自己紹介をすることがネットワーキングでは当たり前に必要なスキルとなる．一般的なアメリカ人であれば，シャイな人でも自己紹介となると意外にハキハキと説明してくれる人が多い．筆者が考えるに，これは日米の教育の差ではないかと思う．アメリカでは，子供のころから転校が多かったり，大人になっても転職が多いので，とにかく自分から自己紹介する機会が多い．また小学生低学年ごろから簡単なプレゼンテーションの方法を学校で教わる．Show and Tellと呼ばれるイベントで，自分の自慢の一品を学校に持ってきて，みんなの前で説明することを，早ければ幼稚園から行う．自分の好きな絵本や買ってもらった玩具，野球のグローブを持ってきたりして，みんなに説明する．小さな子供も道で会ったりすると向こうから「ハイ！」とあいさつしてくることが多い．

大学や専門学校を経て就職活動となると，アメリカでは自己主張を徹底的に行わなければことが始まらない．面接や会社説明でも自分から自分について話すことが必須となる．面接する側もどれくらい積極性があるかを自己主張の度合いで判断するといわれる．アメリカには俗にElevator Talkと呼ばれているものがある．これは設定があり，話を聞いてほしい会社の上役や取引先の人と二人っきりでエレベータに乗る機会があり，先方の行く階に登るまでどれぐらい自分のトークに関心をもってもらえるか？　つまり短い時間で雑談からうまく自分の売り込みをすることを意味する．スピーチのクラスなどでロール・プレイを行って練習することがある．ネットワーキングでの話はまずこのようなElevator Talkに似た状況が多い．とにかく積極的でないとことが始まらない．

☆ ネットワーキングの場

シリコンバレーのエンジニア達のネットワーキングは身近なところから始まる．大学や専門学校の同級生などの知り合いから始まったりするのだと思う．その後就職した場所でまた新たな知り合いができたりする．職場では，会社の中の人々以外にいっしょに仕事をする関連会社，業者，そして顧客などと知り合いになれるチャンスがある．アメリカだと上下関係などの堅苦しいことが少ないので，どこかで知り合いになってそこからネットワーキングができる可能性がある．職場以外では，自分の趣味やオフの活動からネットワークができる．ジムに行ったり，自転車，ロック・クライミングのサークルに入ったりと，さまざまな形がある．またアメリカらしいのが，何らかの教会や宗教のグループでのネットワークだ．日本ではあまりなじみがないかもしれないが，定期的に何らかの礼拝に行くエンジニア達は多い．さらには仕事でカンファレンスや展示会に行くことがあるが，これもネットワーキングの目的も兼ねていると思う．

身近な場所でのネットワーキングが足りないとなると，ネットワーキングが可能なイベントに参加することになる．これらは，特に「ネットワーキング会議」とかいう名前が付いているわけではない．何らかの出し物があり，人が集まるのでネットワーキングにもどうぞご利用ください…といった感じで主催されている．出し物とは，業界著名人によるスピーチ・プレゼンテーションであるとか，パネル・ディスカッションがもっとも一般的だ．出し物のメイン・イベントの前後に飲食する場合があるので，その際に各自で自由に見知らぬ人に声をかけて名刺交換するなりして，ネットワーキングを行う．すべては参加者の意思によるものだから，スピーチとかパネルだけ聞いて帰ってしまう人もいれば，自己紹介をせっせと行ってネットワーキングに力を入れている参加者もいる．

☆ さまざまなタイプのネットワーキング系イベント

シリコンバレーのローカル紙，San Jose Mercury Newsのビジネス欄には毎週さまざまなネットワーキング会議やイベントの告知があり，ざっと平均して毎週20種類以上のイベントが開催されている．イベントの規模もさまざまだし，主催しているグループもさまざまだ．一般的に新聞告知しているタイプは法人格を持っており，NPOがほとんどだ．また，ソフトウェア業界団体をバックにもつSD Forumなどがある．SD Forumはさらに細かく分科会のネットワーク会議がある．業界を活性化するために，サンノゼ市行政当局の支持や大きな企業がスポンサに付いている点が特徴的だ．

そのほかでは，女性エンジニア系，アジア系，中国系，インド系のグループ…つまり人種や国別のグループが存在する．だれでも参加できて，出し物のスピーチの部分は英語で行われ，

そのほかは各自が自由に母国語を使うタイプがある．

日系のグループもいくつかある．それぞれ主旨が違い，一つは一般的に日本のビジネスについて語り合うグループがあり，その一方で日本人による起業をサポートするような団体も存在して，それぞれネットワーキング用のイベントを開催している．小さめのタイプの集まりだと，草の根的な有志の集まりからスタートしたタイプが多い．たとえば，SD Forumの分科会で，モバイル・コンピューティングについてを語り合うグループだ．これらの場合は，出し物がなく，集まって話をすること自体がおもな活動になる．そのほかはインターネット会議で議論を続けたりする．日本的にいうオフ会のような感じだ．たとえば，定期的にピザ屋さんで集まって飲食を共にして，参加した人が自分の飲食代を持ち，リラックスした雰囲気で行われる．

☆ ネットワーク・ミーティングのおもしろ味

ネットワーキングの一つのおもしろ味は日常の仕事以外で付き合いのない会社の人達や，まったく職種の違う人達と出会うことだ．スピーチやイベントの内容によって共通なテーマがあるから，知らない人と話すにも一応は同じテーマに興味があるので，それから話をつなげていけば良い．筆者が定期的に参加するネットワークのイベントは，ベンチャ企業をサポートする弁護士事務所やベンチャ・キャピタルの人達が参加することが多い．ベンチャ企業を起こすことをもくろんでいる人達と知り合うためには，地道に人脈を作らなければならない．彼らにとっては将来への営業活動も兼ねている．しかし，今日明日に何かあるわけではないので地味で気長な活動だ．

同じようにコンサルティング会社や人材斡旋の人達もよくイベントに参加する．業界の情報収集をかねた営業活動だ．テーマによってはまったく違う業界について触れることができる．筆者が利用するネットワーク会議でバイオテックがテーマであったりしても，すべてがわからなくても，行ってみるとなかなか勉強になる．転職を考えているエンジニアなども，じっくりと情報収集できるのでよく利用する．

このように，それぞれの人の目的が何らかの形で一致して集まりができていく．さまざまな業種や企業の上から下まで参加していておもしろい出会いがあると，そのグループ主催のイベントがどんどん人気になり，またそれが人を呼ぶ．どんなにインターネットや電子メールが発展しても実際に会って話ができ，生きた情報交換ができるネットワークのイベントは根強い人気をもつ．最近では，Linked IN，Orkut，Greeなどのビジネス・マンの人脈作り用出会い系サイトが話題になったが，使った人に聞くと，一度登録してしまうと，その後は何もすることがないので放置されてしまうという話だ．やはり定期的に飲食したりせめて会って立ち話ができるとやはり違うようだ．

起業したり，これからスタート・アップを作る人達に，よく先人・先輩の知恵として贈られることばに，「自分の周りを，自分よりもっと賢い人やできる人で固めなさい」というのがある．簡単そうなことばであるが，なかなか奥が深くておもしろいフレーズだと思う．起業を志す人達はまず自分の周りにどんな人がいて，何をしてくれるか考える．シリコンバレーのスタート・アップというと我の強い人達がガリガリ働いているというイメージがあるが，意外とチームで仕事を行っているという意識が強い人が多い．だから，自分の知らないことや不得意なことを補うために，さまざまな人と知り合いになっておくことの大切さをよく理解しているのだと思う．そういう意味で個人それぞれのネットワーキングは大切なスキルだ．

トニー・チン　htchin@attglobal.net　WinHawk Consulting

Column

シリコンバレーの景気についてのアップデート

つい先日(2004年3月頃)までは，3～6か月以上仕事が見つからないエンジニア達が当たり前のように多かった．しかし2004年に入ってからは，徐々に人員を増やしている会社が増えている．シリコンバレーの大手がもっとも多く人員を増やしているらしく，買手市場が続いており，条件があまり良くないが生活のために何でもやろうと考えるエンジニアが増えている．といっても，かなり優秀か，企業が欲しがっているようなスキルがないと難しい．

大手企業は，経費をドンドン削減することによって業績が改善している．筆者の住んでいる近くは，その昔は全盛期のSun MicrosystemsやSilicon Graphicsがオフィスを構えていたところであるが，がら空きのオフィス・ビルが通りの両側に続くところが多い．駐車場を見れば一目瞭然だ．空きビルは，駐車場も空っぽだ．ほぼすべてが貸しビルなので，業績が悪化するとすぐにテナントはいなくなり，周囲は一気に寂しい状態になる．つい2年前にはオフィス・ビルがまったくなく，新しいスタート・アップがドンドン増えていたのが嘘のようだ．やはり大手よりもスタート・アップの方が新しいオフィスを借りたり引っ越したり，人員を増やしたりするので地域が活性化するのは確かだ．また新しいことをやっているため，中で働いているエンジニア達も必死だし，緊張した雰囲気がある．スタート・アップは，やはりシリコンバレーにとっては重要な存在だと痛感する．

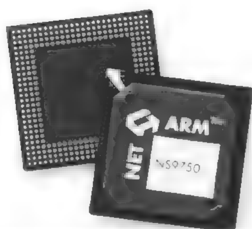
現在もっとも注目されているのはGoogleだ．これは今年中に株式上場するかどうかに皆の関心が寄せられている．ベンチャ企業の上場はテクノロジ系の株価を活性化し，ベンチャ・キャピタル達にも希望を与えるので，周り回ってスタート・アップにも元気を与える．大統領選挙の年は，株式市場が選挙に多いに左右されるので上場には不利な年であるが，Googleの上場の成功を見守る人達は多いと思う．

●32ビットMPU

NS9750

- ARM926EJ-Sコアを採用した，ネットワーク・プロセッサ．
- 10/100Base-T Ethernet，PCI/CardBus，USB（ホストまたはデバイス），I²C，1284（パラレル），シリアル・ポート，GPIO，LCDコントローラなどの周辺機能をワン・チップ化．
- 最大200MHzで動作可能で，DSPコード，Javaバイト・コード・アクセラレータ，MMUに対応．
- 100MHzのメモリ・コントローラを搭載．
- 27チャネルの高速DMAエンジンを搭載．

●価格：下記へ問い合わせ



■ネットシリコン ジャパン（株）
TEL：03-5428-0261　FAX：03-5428-0262

●32ビットRISCコア

SH-2A

- CPUコアの命令処理アーキテクチャの一つであるスーパ・スカラ方式を採用し，最大二つの命令を同時に実行可能．
- データ・バスを命令用とデータ用に分離するハーバード・アーキテクチャを採用することで，命令取り込みとデータ・アクセスの競合が発生しない．
- 最大動作周波数200MHzで，360MIPSの処理性能を実現．
- 単位周波数1MHz当たりの処理性能は，1.8MIPS．
- 16ビット長命令に加え，32ビット長の新規命令やアドレッシング・モードを追加し，FPU関連の21命令を含む，112命令をサポート．
- CPU内部にあるレジスタの情報を格納するための，専用レジスタを内蔵することで，割り込みイベント発生時には，レジスタ情報をハードウェアで高速に専用レジスタに退避できる．

●価格：下記へ問い合わせ

■（株）ルネサステクノロジ
TEL：03-5201-5214

●マイクロコントローラ

ADuC702x

- ARM7TDMIベースのプログラマビリティを特徴とする，32ビットRISCコアとデータ・コンバージョン・テクノロジを集積．
- 最高16チャネルの高速12ビット精度のA-Dコンバータと最高4個の12ビットD-Aコンバータをサポート．
- 10ppm/℃未満のドリフト性能の高精度バンド・ギャップ電圧基準を内蔵．
- 周辺回路として，コンパレータ，PLA，3相PWM発生器を内蔵．
- パワー・ダウン，ウェイク・アップ・モードをサポートしており，3V動作仕様．
- 動作温度範囲は，－40℃～+85℃，+105℃および+125℃．
- FA用センサ，光ネットワーク用送信機，自動車の車体制御と監視を大幅に簡素化するなどの応用が考えられる．

●価格：$4.55～$12.80（1,000個時）

■アナログ・デバイセズ（株）
TEL：03-5402-8268

●8ビット・マイコン

F²MC-8FXファミリ

- 新開発の回路線幅0.35μmでの低リーク電流技術により，消費電流1μAを実現．
- 最大動作周波数は10MHz．
- アドレスとデータの並列処理を可能にする内部バスのパイプライン化，処理命令を実行する前に蓄積しておくことで処理効率を高めるプリフェッチ・バッファを搭載．
- 「MB95F108/MB95F108H」は，独自のデュアル・オペレーション・フラッシュ・メモリを搭載したことにより，外付けメモリ部品を削減でき，実装面積を縮小．
- 評価用チップ「MB95F100-101/MB95F100-102」および評価用ハードウェア・ツール「MB2146-09」を提供．
- ディジタル・スチル・カメラやビデオ・カメラなどの操作部の制御から，給湯器や炊飯器の制御まで幅広い用途に適応．

●サンプル価格：
¥750（MB95F108/MB95F108H）
¥35,000（MB95FV100-101/MB95FV100-102）
¥30,000（MB2146-09）

■富士通（株）
TEL：042-532-1397

●セット・トップ・ボックス用LSI

QAMi5516

- アナログTVでディジタル・ケーブルTVの視聴を可能にする，単一チップ・ソリューション．
- ディジタル・ケーブル用STBに適しており，高性能CPUと拡張グラフィクス機能を搭載．
- QAMケーブル信号モジュレータ機能とディジタル信号を複合するMPEGビデオ・ストリーム・デコーダ機能を搭載．
- チップセットのコストを削減し，プリント回路基板設計のシンプル化と実装面積の縮小を実現．
- ケーブル放送用STBの各種要件を満たし，主要なコンディショナル・アクセスやミドルウェア製品をサポート．

●価格：下記へ問い合わせ



■STマイクロエレクトロニクス（株）
TEL：03-5783-8340　FAX：03-5783-8216

●A-Dコンバータ

LTC1403

- ミッシング・コードのない14ビット，サンプリング・レート2.8Mbpsの，高速データ収集設計向けのシリアルA-Dコンバータ．
- 73.5dBのSINAD，±4LSBのINLというAC/DC性能を提供し，産業用アプリケーションと高速イメージング・アプリケーションのいずれにも適する．
- 3V電源を使用して，4Mbpsでサンプリングを行う場合の消費電流は5mA．
- 無変換時の消費電流は，内蔵2.5Vリファレンスがアクティブである NAPモードで3.3mW，すべてがパワー・ダウンするスリープ・モードで6μWに低減．
- 80dBの同相入力除去比により，ソースから差動で信号を測定することによって，グラウンド・ループと同相ノイズを除去できる．

●サンプル価格：¥435～（1,000個時）



■リニアテクノロジー（株）
TEL：03-5226-7291　FAX：03-5226-0268

●ビデオ・アンプ

TSH8xファミリ

- 広帯域，GNDまでの入力動作，レール・ツー・レール出力動作の小型パッケージ・アンプ．
- RGB信号ドライブおよびスイッチなど，信号増幅と信号処理が要求される量産ビデオ・アプリケーションの要件を満たす．
- TSH80はシングル，TSH81はスタンバイ付きシングル，TSH82はデュアル電圧フィードバックのOPアンプ．
- 100MHzの利得/帯域幅製品で，スルー・レートは100V/μs．
- 出力側は，150Ωの負荷を駆動できるように最適化されている．

●サンプル価格：TSH80 ¥60（1,000個時）
TSH81 ¥60（1,000個時）
TSH82 ¥80（1,000個時）



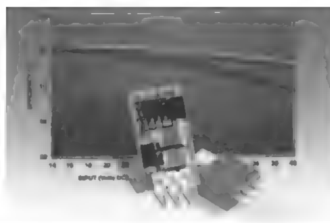
■STマイクロエレクトロニクス（株）
TEL：03-5783-8240　FAX：03-5783-8216

●DC-DCコンバータ

78SRシリーズ

- 3端子レギュレータ互換で直接代替可能な，TO-220SIPパッケージのステップダウン・レギュレータ．
- 260kHzのスイッチング周波数で95%の高効率を実現し，－40～70℃動作範囲でヒートシンクなしで使える製品．
- 出力電圧/電流が，3.3V/0.5A，5V/0.5A，12V/0.4Aの3モデルを用意．
- 入力電圧範囲は，3.3/5Vモデルが7.5～36V，12Vモデルが15～36V．
- 静止電流は3mAで，逆極性接続保護や入出力コンデンサを内蔵しているため，外付け部品が不要．
- 出力電圧精度は，±1.5%．

●価格：¥850（10～24個時）



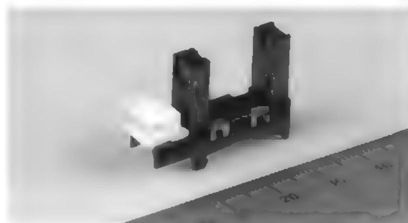
■デイテル（株）
TEL：03-3779-1031　FAX：03-3779-1030

●透過型ワイドギャップ・フォト・インタラプタ

LG285シリーズ

- 発光素子に高出力赤外発光ダイオード，受光素子に高感度フォトICを用い，13mmのギャップを持つ．
- コネクタ接続タイプで，指定コネクタの取り付けが可能．
- 基板取り付けはスナップ・インが可能で，4種類の基板厚みに対応．
- 位置検出特性は，X方向，Y方向ともに検出可能．
- 入光時ハイレベル出力（ダーク・オン），遮光時ハイレベル出力（ライト・オン）の選択が可能．
- 外乱光による誤動作を防止するために，可視光カット・フィルタの取り付けが可能．

●価格：下記へ問い合わせ



■コーデンシ（株）
TEL：0774-23-7113　FAX：0774-20-3916

●モータ駆動モジュール

IR3101

- ゲート駆動ICと出力パワーMOS FET 2個を内蔵．
- 最大400Wの冷蔵庫用のコンプレッサ，ファン，ポンプの単相，2相，3相モータを駆動するハーフブリッジ・インバータの設計を簡素化．
- 各パワーMOS FETは，回復時間が高速なボディ・ダイオードを装備．
- 雑音耐性と損失を改善する低*di/dt*のゲート駆動機能を提供．
- MOS FETの伝搬遅延時間が整合しているため，スイッチング特性を合わせやすく，出力の歪みを低減．
- モジュールの最大入力電圧は500V，最大消費電力は5.8W．
- 完全絶縁型のミニSIPパッケージを使用しており，すべてのピンにESD保護機能が付いている．

●サンプル価格：¥650

■インターナショナル・レクティファイアー・ジャパン（株）
TEL：03-3983-0086　FAX：03-3983-0642

●OPアンプ

OPA734/OPA735 OPA2734/OPA2735

- OPA734はシングルでシャットダウン・モード付き，OPA735はシングルでシャットダウン・モードなし，OPA2734はデュアルでシャットダウン・モード付き，OPA2735はデュアルでシャットダウン・モードなしのOPアンプ．
- OPA734/OPA735は，オート・ゼロ技術を使用して5μVの低オフセット電圧，および時間経過や周囲温度の変化に対して，0.05μV/℃のオフセット・ドリフト特性を提供．
- 750μAの無信号時動作電流，200pAのバイアス電流，1.6MHzの帯域幅，正負両電源電位の50mV以内までのレール・ツー・レールの出力振幅特性などの特徴を持つ．
- ＋2.7V～＋12Vの単一電源，または±1.35V～±6Vの正負両電源で動作．
- 特性は，－40～＋125℃の温度範囲で規定される．

●価格：OPA734/OPA735 ¥200（1,000個時）
OPA2734/OPA2735 ¥260（1,000個時）

■日本テキサス・インスツルメンツ（株）
FAX：0120-81-0036

●ETC用RFモジュール

UGHP6シリーズ

- 高機能基板の採用，両面実装などにより，外形サイズ22.4×22.4×4.6mmを実現．
- 従来品と比較して，約70%の小型化（容量2.31ml）を達成．
- 車載用SMD規格に対応し，セット側で簡易に表面実装が可能．
- アンテナ内蔵の2ピース・タイプ，アンテナを外付けする3ピース・タイプを用意．
- 送受信周波数間隔は40MHz，チャネル間隔は10MHz，チャネル数は2チャネルをサポート．
- 伝送速度は1.024Mbpsで，電源電圧は3.3/5.0V．

●サンプル価格：¥20,000

■アルプス電気（株）
TEL：03-5499-8154

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

NEWPRODUCTS

●車載用AM/FMチューナ

TDGB2シリーズ

- ベース・バンド部のDSPに対応した，フロント・エンド・チューナ．
- 独自の高周波回路技術により，送電線により発生するノイズを制限する回路を内蔵．
- 不安定な電波条件の中でも，最大の性能を発揮する機構を搭載．
- アンテナDAA（Digital Auto Alignment）を内蔵．
- I²CバスによるAGC（Automatic Gain Control）スレッショルド設定が可能．
- 高感度，低歪み特性を実現．
- 優れた位相雑音特性を実現．
- 全世界の周波数に対応．
- 52.0×42.0×13.0mmの小型化を実現．
- 電源電圧は8.5/5.0Vで，消費電流は50mA（8.5V）/35mA（5.0V）．

●サンプル価格：¥20,000

■アルプス電気（株）
TEL：03-5499-8154

●検流増幅器

T-IVA001H

- 微小電流を検出し，電圧変換を行う．
- チャネル数は，1チャネル．
- 電流電圧変換率は，10G（V/A），100G（V/A）の2レンジをサポート．
- 変換率精度は，±1%以内．
- 立ち上がり応答200μsの，周波数特性を実現．
- 雑音出力は，100Gレンジ/フィルタ10kHz時に約2pArms，フィルタ1kHz時に約10fArms．10Gレンジ/フィルタ10kHz時に約2pArms，フィルタ1kHz時に約40fArms．
- 最大出力電圧は約±5Vp，最大出力電流は±10mA．
- オフセット調整は，前面パネルより調整可能．調整幅は，±2pA以上．

●価格：¥48,300



■（株）タートル工業
TEL：029-843-0045　FAX：029-843-2024

●無線LANアダプタ

SE-50VoIP

- IP電話機能を搭載した無線LANアダプタとして利用でき，IP電話の音声およびパソコンのIP電話機能を搭載した無線LANアクセス・ポイントへの無線伝送を行う．
- 無線LANブロードバンド・ルータを親機，SE-50VoIPを子機として，IP電話システムの構築が可能．
- UPnPクライアント機能に対応しているため，個々にIP電話番号を取得すれば，LAN内からダイレクトに発信/着信が可能．
- 無線LANアクセス・ポイントとの通信形態は，ブリッジ・モード，ルータ・モードのいずれかから選択可能．
- DHCPサーバ機能により，有線LAN側のパソコンのIPアドレス設定は自動で実行される．
- 5.2GHz，54Mbps（IEEE802.11a準拠）および2.4GHz，54Mbps/11Mbps（IEEE802.11g/IEEE802.11b準拠）の無線通信を実現．
- セキュリティとして，次世代標準化暗号方式OCB AES（128ビット）を搭載．

●価格：オープン価格

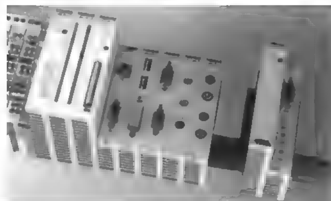
■アイコム（株）
TEL：06-6792-4949

●モジュラ型エンベデッドPC

CX1000

- ドイツのベッコフ社製のエンベデッドPCの標準モジュール．
- MMX Pentium互換（266MHz）のCPUと16Mバイト・フラッシュ・メモリ，32MバイトRAMの2種類の内蔵メモリで構成．
- コンパクト・フラッシュにより，最大1Gバイトまで増設することが可能．
- EthernetとRS-232-Cインターフェースを標準で搭載．
- I/Oコンポーネットは，CXファミリのラインナップからプラグイン方式で追加することが可能．
- CPUはファンレスで，ヒートシンクを標準装備．
- 電源モジュールは，CX1100を使用．

●価格：下記へ問い合わせ



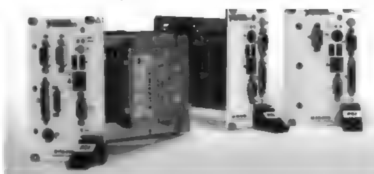
■（株）ケーメックス
TEL：03-5295-3111　FAX：03-5295-3123

●PXI組み込み型コントローラ

NI PXI-8180シリーズ

- PXI-8186は，Pentium4プロセッサ（2.2GHz）を搭載し，USB2.0を内蔵．PXI-8185はCeleronプロセッサ（1.2GHz），PXI-8184はCeleronプロセッサ（850MHz）を搭載．
- 手持ちのPCからEthernet経由で，PXIを遠隔操作することが可能．
- PXI-8186の演算処理能力により，高度な解析が可能となり，テスト時間の短縮を実現．
- PXI-8185/PXI-8184を利用することで，モジュール式計測用の小型で高性能なPCプラットホームの作成が可能．
- ハード・ドライブ，Ethernet，USB，GPIB，シリアルなどの周辺機器を単一モジュールに組み入れることで，システム統合が可能．

●価格：¥575,000～（PXI-8186）
¥431,000～（PXI-8185）
¥287,000～（PXI-8184）



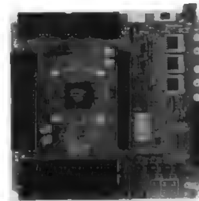
■日本ナショナルインスツルメンツ（株）
TEL：03-5472-2970　FAX：03-5472-2977

●Cycloneブレッド・ボード

CSP-026シリーズ

- アルテラ社のFPGAを実装した，評価用基板の完成品．
- 直付けタイプとソケット・タイプの2タイプを用意．
- 233本のI/Oピンを，GPIFコネクタに引き出している．
- 8層基板を使用し，電源回路，リセット回路，クロック源，ISP可能コンフィグレーションROMを実装．
- コネクタはASモード用とJTAG用の2種類で，それぞれByte Blaster IIやUSB Blaster，BL3，BLKIT（REV2）に対応．
- コンフィグレーションROMは，10,000回以上書き換えが可能なEPCS4を使用し，Byte Blaster IIやBL3などで書き込みが可能．

●価格：下記へ問い合わせ



■（有）ヒューマンデータ
TEL：072-620-2002　FAX：072-620-2003

●USBリモート・ハブ

AnywhereUSB

- USB over IP技術を利用したリモート・ネットワーキング機器.
- USBデバイスをLAN，あるいは無線LAN上のどこにでも設置でき，接続のためのPCは不要.
- 5個のUSBポートを提供しており，各ポートはローカルなUSBポートと同じプラグ&プレイを提供.
- ホストPCを遠隔地に設置できるため，セキュリティを強化でき，システムの冗長化を実現.
- 通信は，TCP/IP接続経由で行える.
- ケーブル長限界の5mを越えられる.

●サンプル価格：¥65,100



■（株）昌新
TEL：03-3270-5926　FAX：03-3245-1695

●デバイス・サーバ

WiPort

- 10.2×32.5×33.0mmの筐体に，CPU，メモリ，無線LAN（IEEE802.11b対応），Ethernetチップおよび周辺回路などを内蔵.
- OS，TCP/IPプロトコル，Webサーバ，メール発信機能，暗号などのソフトウェアをサポート.
- 同製品を搭載した各機器は，個別のホームページを機器内に持つことができるため，遠隔地からの各機器のモニタ，保守などが可能.
- 通信プロトコルの移植作業やソフトウェアのライセンス契約は不要.
- インターネットに接続が困難であった機器や，有線ネットワークに接続が困難であった機器のネットワーク接続を可能にする.
- ディジタル家電機器などを始め，ホーム・ネットワークやオフィス，工場環境などの分野に適応.

●価格：
下記へ問い合わせ



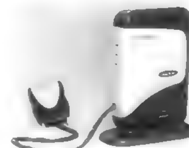
■日本ラントロニクス（株）
TEL：03-3780-7025　FAX：03-3780-7026

●開発環境

E200F エミュレータ

- 個々の関数プログラムの処理時間を測定，一覧表示するリアルタイム・プロファイラ機能を搭載することで，プログラムを停止することなく，リアルタイムでの測定が可能なSH-4A/SH4AL-DSP向けエミュレータ.
- 個々の関数の実動作状態における正確な処理時間を把握でき，規定の時間内で処理できているかどうかを判断できる.
- プログラム空間上の指定範囲は最大12Mバイトを設定できる.
- マイコンに搭載される周辺I/Oモジュール対応のアナライザ機能を搭載することで，プロトコルを解析しながらプログラムの停止や実行と連動でき，デバッグ効率の向上が図れる.

●価格：¥360,000（メイン・ユニット）
¥240,000（トレース・ユニット）
¥240,000（拡張プロファイル・ユニット）



■（株）ルネサスソリューションズ
TEL：03-5201-5022

●セキュリティ製品

RSA USB トークン 6100

- パスワードやディジタル証明書を，USB型のリーダに格納したJavaカードのスマート・カード・チップ上に保管する，USB型のハードウェア・コンテナ.
- Windowsログオン用のIDとパスワードのペアを，最大3組まで格納可能.
- パスワードの盗難，キー・ロガーによるパスワードの収集を予防できる.
- ディジタル証明書を最大三つまで格納可能.
- ディジタル証明書をパソコンに格納する場合と比較して，安全面に優れており，ほかのパソコンでも自分のディジタル証明書を利用できるなど，可搬性を実現.
- 「RSA SecurIDソフトウェアトークン」のシード機能を，最大三つまで格納できるため，強力な二要素認証を実現.

●価格：¥880,000/100ライセンス



■RSAセキュリティ（株）
TEL：03-5222-5230

●オシロスコープ

TDS6000Bシリーズ

- 周波数帯が8GHzのTDS6804B型と6GHzのTDS6604B型を用意.
- TDS6804B型は，7GHzのアナログ帯域幅とDSPによる8GHz帯域幅の切り替えが可能．DSP動作時で，35ps（20～80%）の立ち上がり時間を実現.
- 4チャネル同時20Gspsでのアクイジションが可能で，各チャネルは最高32Mポイントまで同時に記録することが可能.
- 32Mポイントのレコード長は，フルサンプル・レート時で，16msのデータ補足が可能.
- 独自のピン・ポイント・トリガ機能のAおよびBのデュアル・トリガ・システムを使用することにより，1,400種類以上のトリガ・コンビネーションが可能.
- 1.5ps以下と，トリガ・ジッタを大幅に低減.

●価格：下記へ問い合わせ

■日本テクトロニクス（株）
TEL：03-6714-3464　FAX：03-6714-3663

●iSCSIソリューション

SANmelody

- 低価格なPCサーバを，最小のコストで最大の性能をもつ，高度なストレージ・サービスを提供するプラットホームとなるハードウェアとして利用可能にする.
- iSCSI構成において，安価なEIDEおよびシリアルATAドライブを使用し，高速なCPU，キャッシュ・メモリ，標準Ethernetカードの利点を生かして，アプリケーションのI/Oニーズに応える.
- TCPオフロード・エンジン，iSCSIホスト・バス・アダプタなどの高価なハードウェアを使わずに，パフォーマンス全体を向上させることが可能.
- ソフトウェアのアップグレードにより，ポイント・イン・タイム・スナップショット，自動割り当て，リモート・レプリケーションなどの機能の追加が可能.

●価格：下記へ問い合わせ

■データコア・ソフトウェア（株）
TEL：03-3511-1711　FAX：03-3511-1712

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX:（03）5395-2127，E-mail: mngnews@cqpub.co.jp　（編集部）

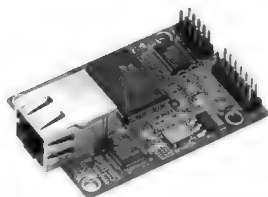
NEWPRODUCTS

●Ethernetインターフェース・モジュール

NE-4110S

- TCP/UDPなどのEthernet接続ソリューションを提供しているため，RS-232-Cシリアル・インターフェース機器を高速のネットワークに対応させることが可能．
- 57×49mmのコンパクト・サイズを実現しているため，シリアル・インターフェース機器に簡単に組み込みができ，10/100MbpsのEthernetと直接接続が可能となる．
- 10/100Mbps Ethernetの自動検出インターフェース機能をもっている．
- コンフィグレーションは，Windowsユーティリティ，Webブラウザ，Telnetコンソール，シリアル・コンソールで行われる．

●価格：下記へ問い合わせ



■山下システムズ（株）
TEL：03-5700-2121　FAX：03-5700-0076

●組み込み開発キット

EDK バージョン6.2i

- マニュアルで設計していたステップを自動化し，バグの検出と修正を迅速に行うようにハードウェア/ソフトウェア統合デバッグを実現することにより，組み込みシステム開発を高速化．
- Platform Studio EDKの機能強化により，ハードウェア・デザイナとソフトウェア・デザイナの両方が共通デザイン環境を用いて，PowerPCまたはMicroBlazeをターゲットとした開発が可能．
- 一連のソフトウェアは，ターゲット・ボード，プロセッサおよびIPをベースとして，VxWorks，MontaVista Linux，Xilinx MicroKernelのためのBSPを生成する能力を拡大．
- プログラマブル・プラットホームのデバッグ能力とハードウェア/ソフトウェア・ドメインにまたがる問題を特定するための予見能力を提供する，ユニークなPlatform Debug機能を搭載．

●価格：下記へ問い合わせ

■ザイリンクス（株）
TEL：03-5321-7740　FAX：03-5321-7762

●GUI開発ツール

Stingray Studio 2004

- 米国Rogue Wave Software社が開発した，GUIアプリケーション開発の細部を処理するために設計されたMFC，ActiveX，Microsoft.NET用コンポーネント製品．
- 完全なソース・コードが含まれているため，要件に合わせたコンポーネントの柔軟なカスタマイズが可能．
- Objective Grid，Objective Toolkit，Objective Edit，Objective Chart，Objective Viewsの五つの製品が含まれる．
- Objective Gridで作成したグリッドとExcelの相互運用で，Microsoftのオートメーションに基づいた実装方法を採用．
- マルチ・モニタのサポートにより，Objective Toolkitを使用して作成したアプリケーションのマルチ・モニタ環境での実行が可能．
- Objective GridやObjective Viewsの機能を，Visual C#開発環境へエクスポートすることが可能．

●価格：¥307,650

■エクセルソフト（株）
TEL：03-5440-7875 FAX：03-5440-7876

●SH7770用描画ソフトウェア

グラフィックスソフトウェアエンジン

- SH7770が内蔵する2D/3Dグラフィックス・エンジン用のAPIを提供するプログラムで，グラフィックス・エンジンの性能を最大限に引き出せるよう最適化されている．
- APIの使用により，複雑なプログラムを作成することなく，高品質で高速な表示画面を短期間で実現．
- 2Dおよび3Dの各グラフィックス・エンジンを効率よく，高速に並列動作させることが可能．
- 描画の最大性能は，250Mピクセル/sを達成し，従来と比較して，ライン描画2倍，矩形描画4倍，多角形描画4倍，鳥瞰地図描画40倍の処理速度の向上を実現．
- 車載情報端末向けのOS「Microsoft Windows Automotive v.4.2」上で動作．

●価格：下記へ問い合わせ

■（株）ルネサスソリューションズ
TEL：03-5201-5120

●ソース・コード・アラインメント・ツール

Jindent3.51

- ドイツのSoftware＆Solution社が開発した，Java環境で動作する多機能なソース・コード管理ユーティリティ．
- 統一されていないソース・コード記述ルールを，短時間で一致させることが可能．
- 複数のJavaソース・コードを比較する場合，ソース・コード記述ルールを作成し，ルールをファイルに適用した後に比較を行い，修正，変更部分の把握を短時間で行うことが可能．
- インデント幅や括弧の記述を統一することで，ソース・コード記述ミスを未然に防止することが可能．
- JBuilder，Eclipse，Visual Cafeなどの，Java IDEへの統合をサポート．
- バッチによる処理，APIからの使用，テキスト・エディタへの統合などをサポート．

●価格：¥21,000（Basic）
¥31,500（Gold）

■エクセルソフト（株）
TEL：03-5440-7875 FAX：03-5440-7876

●検証用ツール

Board Support Toolkit

- MontaVista Linux Professional Edition上での動作に対する認証付きのカスタムLSPを作成し，利用可能にするためのツールを提供．
- LSPをパッケージ化するためのツール，安定性を検証するためのテスト・スイート，包括的なドキュメンテーションを含んでいる．
- BST Creation Toolは，MontaVista Linux互換のフォーマットで，カーネル・ポートや関連するドライバをパッケージ化することを可能にしている．
- BST Certification Suiteは，MontaVista Linuxに含まれる広範囲なアプリケーション・ソフトウェアと互換性を持ち，LSPのレイアウト，アーキテクチャ，コンテンツを認証．

●価格：下記へ問い合わせ

■モンタビスタソフトウェアジャパン（株）
TEL：03-5469-8840　FAX：03-5469-8801

●汎用ファイル・システム

OSE組込ファイル・システム

- プログラム・モジュールの格納や種類の異なったストレージ・デバイス間のデータ転送に利用可能.
- コンパクトでありながら，単一CPUもしくは分散型プラットホーム上で，多数の異なったストレージ・デバイスを使用したシステムをサポート.
- APIは，標準のUNIX/C関数コール(POSIX 1003.1またはANSI C)，またはOSEメッセージを使用.
- システム管理コマンド一式は，保守および監視機能を提供.
- モジュラ・アーキテクチャを採用し，新しいメディア・タイプとファイル形式に対応できる.
- Cランタイム・ライブラリ，ファイル・システム・サーバ，ファイル・マネージャ，デバイス・ドライバなどのコンポーネントで構成.

●価格：下記へ問い合わせ

■エニア・エンベデッド・テクノロジー(株)
TEL：03-5207-6167　FAX：03-5207-6169

●ネットワーク・ドライバ

GR-WinNET

- Windows上で，組み込み向けネットワーク・プロトコル・スタックを動作させることが可能.
- 組み込み向けネットワーク・プロトコル・スタックを使用したWindows上のアプリケーションで，上位ネットワーク・プロトコルの開発が可能.
- 複数のシミュレーション・アプリケーションと通信が可能.
- Windows上のネットワーク・アプリケーションとの通信が可能.
- Windowsのルーティング機能を使用し，外部機器との通信が可能.
- ARP/RARP/IPなどの，低レベル・パケット送受信をサポート.
- 各種ネットワーク・プロトコル・スタックに対応.
- サンプル・ドライバを標準で添付.

●価格：下記へ問い合わせ

■(株)グレープシステム
TEL：045-222-3761 FAX：045-222-3759

●言語変換ツール

Verilog2SystemC

- Verilog-HDLで記述されたRTLモデル，ビヘイビア・モデル，テスト・ベンチなどを，SystemC言語のデザインへ自動変換.
- SystemCを用いることで，高速な検証が可能.
- Verilog-HDL資産を活用することにより，スムーズな設計環境のシフトが可能.
- 一部のシミュレーション構文を除き，Verilog-HDLのフル構文をサポート.
- マクロ対応によるゲート・レベルの変換をサポート.
- SystemCの最新バージョン2.0.1に対応.
- オプションで，Four-valued Logic(0,1,X,Z)，Two-valued Logic(0,1)をサポート.
- Verilog-HDL回路の遅延情報を意識した回路の変換が可能.
- 動作環境は，Red Hat Linux8.0, Windows, Cygwinに対応.

●価格：¥1,8000,000

■(株)礎デザインオートメーション
TEL：03-6762-1471　FAX：03-6762-1472

●USBホスト・コントローラ用ドライバ

Windows CE用USBホスト・ドライバ

- EZ-HostおよびEZ-OTG組み込みホスト・コントローラ用のWindows CEドライバ.
- Windows CE.NET 4.2，およびWindows Mobile 2003をサポートしており，既存のUSBスタックとWindows CEで提供されるクラス・ドライバと連動.
- 既存USB周辺装置と通信するUSBホスト，OTGベース製品の利用が可能.
- シリコン，ドライバ両面において，OTG規格を完全にサポートし，認証試験済み.
- EZ-HostおよびEZ-OTGベース製品へ接続可能なデバイス数が，大幅に拡大する.

●価格：下記へ問い合わせ

■日本サイプレス(株)
TEL：03-5371-1921　FAX：03-5371-1955

●OS API

Nucleus POSIX

- POSIX APIサポートを持つNucleus以外のRTOS向けアプリケーションを，Nucleus RTOSへと移植が可能.
- POSIX APIの普及率が高い，UNIXをバックグラウンドに持つことで，新しい環境の学習に時間を費やすことなく，慣れ親しんだAPIにおいて組み込みアプリケーションの開発が可能.
- POSIX APIの知識をもつエンジニアを採用することで，人材育成時間を短縮できる，OSの移行が必要な場合でも，POSIX APIをサポートするOS間では簡単にアプリケーションの移植が行えるなど，企業エンジニアリング投資の効率化を図れる.

●価格：下記へ問い合わせ

■メンター・グラフィックス・ジャパン(株)
TEL：03-5488-3041　FAX：03-5488-3032
URL：http://www.acceleratedtechnology.jp/

●統合開発環境

MULTI

- 米国Green Hills Software社が，MIPS Technologies社の24Kコア・ファミリをサポート.
- MIPS32/64アーキテクチャ向けに最適化された広範囲なツール・セットは，シミュレータやハードウェア検証プローブ，幅広いアプリケーションにわたるオペレーティング・システム・サポートを統合.
- ソース・レベル・デバッガ，パフォーマンス・プロファイラやランタイム・エラーチェッカのほかに，最適なコードを自動チューニングするCode Balance，リアルタイム・イベント・アナライザ，バージョン・コントロール・システム，グラフィカル・プログラム・ビルダや複数言語をサポートするエディタを提供.
- ターゲット・ハードウェアがなくても，開発コードのプログラムを可能にする命令セット・シミュレータを提供.

●価格：下記へ問い合わせ

■(株)アドバンスド・データ・コントロールズ
TEL：03-3576-5351 FAX：03-3576-1772
E-mail：sales@adac.co.jp

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX:(03)5395-2127，E-mail：mngnews@cqpub.co.jp　(編集部)



INFORMATION
海外・国内イベント/セミナー情報



海 外 イ ベ ン ト

6/6-9	**2004 Taiwan Semiconductor Fair** Taipei World Trade Center Exhibition Hall, Taipei, Taiwan Semiconductor Industry Association http://www.semichips.org/eve_event.cfm?ID=153
6/7-10	**Sensors Expo & Conference Spring 2004** Cobo Exhibition/Conference Center, Detroit, MI, USA Advanstar Communications http://www.sensorsexpo.com/
6/8-10	**Wireless Connectivity World** Amsterdam RAI, Amsterdam, Netherlands IBC Telecoms & Media Group http://www.wiconworld.com/
6/28-7/1	**JavaOne Conference in San Francisco** Moscone Center, San Francisco, CA, USA Sun Microsystems http://java.sun.com/javaone/

国 内 イ ベ ン ト

5/31-6/1	RSA Conference 2004 Japan 赤坂プリンスホテル(東京都千代田区紀尾井町) RSA Conference 2004 Japan 実行委員会 http://www.medialive.jp/events/rsa2004/
6/1-4	Agilent Measurement Forum 2004 東京コンファレンスセンター・品川(東京都港区港南) アジレントテクノロジー(株)電子計測本部 http://www.agilent-epsg-j.com/amf04/
6/2-4	ビジネスショウ OSAKA 2004 インテックス大阪(大阪府大阪市住之江区) (社)日本経営協会 http://www.noma.or.jp/bsosaka/
6/2-4	JPCA Show 2004 東京ビッグサイト(東京都江東区有明) (社)日本プリント回路工業会 http://www.jpca.jp/
6/2-4	LinuxWorld Expo/Tokyo 2004 東京ビッグサイト(東京都江東区有明) (株)IDGジャパン http://www.idg.co.jp/expo/lw/
6/3-4	TOPPERSコンファレンス 学士会館(東京都千代田区神田錦町) NPO法人 TOPPERSプロジェクト http://www.toppers.jp/
6/9-11	光ナノテクフェア 2004／2004光計測シンポジウム 日本光学測定機工業会 パシフィコ横浜(神奈川県横浜市西区) http://www2.ocn.ne.jp/~joma/
6/10	STORAGE NETWORKING WORLD/Tokyo 2004 Spring 大手町サンケイプラザ(東京都千代田区大手町) (株)IDGジャパン http://www.idg.co.jp/expo/snw/
6/11	Mobile & Wireless World/Tokyo 2004 大手町サンケイプラザ(東京都千代田区大手町) (株)IDGジャパン http://www.idg.co.jp/expo/mww/index.html
6/22	Java WORLD DAY 2004 東京全日空ホテル(東京都港区赤坂) (株)IDGジャパン http://www.idg.co.jp/jwday/index.html
6/28-7/2	NetWorld+Interop 2004 Tokyo 幕張メッセ(千葉県千葉市美浜区) NetWorld+Interop 2004 Tokyo 実行委員会 http://www.interop.jp/
6/30-7/2	第14回 フラットパネルディスプレイ製造技術展 東京ビッグサイト(東京都江東区有明) リード エグジビション ジャパン(株) http://web.reedexpo.co.jp/ftj/

開催日，イベント名，開催地，問い合わせ先の順

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

セ ミ ナ ー 情 報

Javaによるプロセス指向と並列プログラミング～高速性，小さなコードサイズを活かしたJAVAの新しいアプリケーション開発方法～
開催日時　：6月1日(火)
開催場所　：オームビル(東京都千代田区)
受講料　：55,335円
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831，
E-mail：seminar@triceps.co.jp
http://www.catnet.ne.jp/triceps/sem/040601a.htm

USBの基礎と応用
開催日時　：6月3日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版社エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

第2回 GPL/LGPLセミナ
開催日時　：6月3日(木)～4日(金)
開催場所　：(株)イーエルティ 西日本営業所(大阪府大阪市淀川区)
受講料　：60,000円(税抜)
問い合わせ先：(株)イーエルティ，E-mail：seminar@emblit.co.jp
http://www.emblit.co.jp/event.html

TCP/IPによるI/O制御の実際～Ethernetを利用した組み込み機器の設計
開催日時　：6月4日(金)～5日(日)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：25,000円
問い合わせ先：CQ出版社エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

トランジスタ技術テクノロジ・セミナー 実験で学ぶパワー・エレクトロニクス
開催日時　：6月8日(火)
開催場所　：パシフィコ横浜アネックスホール(神奈川県横浜市西区)
受講料　：36,750円(無料セッションあり)
問い合わせ先：CQ出版社TSE事務局，☎(03)5395-1465，FAX(03)5395-3911
http://it.cqpub.co.jp/tse/tr200406/

VHDL入門
開催日時　：6月10日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版社エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

ビギナのための電子工学講座
開催日時　：6月12日(土)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：3,000円
問い合わせ先：CQ出版社エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

RF-IDの標準化動向とアプリケーション解説
開催日時　：6月17日(木)～18日(金)
開催場所　：オームビル(東京都千代田区)
受講料　：65,940円
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831，
E-mail：seminar@triceps.co.jp
http://www.catnet.ne.jp/triceps/sem/040617a.htm

C言語ベースのシステムLSI設計
開催日時　：6月18日(金)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版社エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

自動車向けソフトウェア開発手法
開催日時　：6月21日(月)
開催場所　：オームビル(東京都千代田区)
受講料　：55,335円
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831，
E-mail：seminar@triceps.co.jp
http://www.catnet.ne.jp/triceps/sem/040621a.htm

組み込みセキュリティソリューションセミナー
開催日時　：6月23日(水)
開催場所　：クイーンズフォーラム会議室(神奈川県横浜市西区，クイーンズタワーB 7F)
受講料　：無料
問い合わせ先：(株)グレープシステム，E-mail：info@gr.grape.co.jp
http://www.grape.co.jp/seminar.html

磁界シミュレータでわかる高周波技術
開催日時　：6月24日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：12,000円
問い合わせ先：CQ出版社エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

Linux GUIプログラミング
開催日時　：6月28日(月)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：46,000円
問い合わせ先：(株)エイチアイ ICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/linux/clinuxgui.asp

Linux-zaurusプログラミング
開催日時　：6月29日(火)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：46,000円
問い合わせ先：(株)エイチアイ ICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/linuzau/linux_zaurus.asp

読者の広場

Interface への声



2004年4月号特集「組み込みシステムの世界へようこそ！」に関して

▷先月も組み込みシステムの仕事で超過勤務時間が100時間を越えた．メンタルな面からの開発手法は今後考えていかなければならない．超過勤務が増えるのは本来頭脳労働たる開発を肉体労働に還元しようとして無理に工数を出すからだろう．頭脳労働の生産は必ずしも時間に比例しないし，個人間の能力差も大きい．（過労士）

▷別冊付録『ISAバス＆Cバス活用ハンドブック』は，そんな昔ではないはずなのに，懐かしさが漂って（?）くるのは，この業界の移り変わりが激しいからでしょうか．付録では74TTLが並んでましたが，もし今，基板を作るなら，PLDでも使って1チップでしょうか．今後万が一，ISAバスやCバスの仕事が来たとき困らないよう，資料として取って置こうと思います．

（元PC98ユーザ）

アンケートの結果

興味のあった記事（2004年5月号で実施）

①第1章 組み込みシステムとは何か
②第2章 社会のインフラ――組み込みOS
③第4章 組み込み向けクロス開発環境の構築
④組み込みプログラミング・ノウハウ入門（第14回）
⑤第3章 実例で学ぶハードウェアのデバッグ
⑥組み込みLinuxのROM化の実例
⑦プログラミングの要（第11回）
⑧フリーソフトウェア徹底活用講座（第14回）
⑨第5章 CodeWarriorを使用した組み込み開発
⑩開発技術者のためのアセンブラ入門（第25回）
⑪移り気な情報工学（第38回）
⑫シニアエンジニアの技術草子（参拾八之段）
⑬別冊付録ISAバス＆Cバス活用ハンドブック
⑭iSCSI技術のストレージ製品への応用
⑮やり直しのための信号数学（第23回）
⑯IPパケットの隙間から（第62回）
⑰ハッカーの常識的見聞録（第42回）
⑱TOPPERSで学ぶRTOS技術（第6回）
⑲Linuxで使うOMAP DSP部のソフトウェア開発
⑳TMS320C6713搭載DSPスタータ・キットを使ったC++によるDSPオブジェクト指向プログラミング（第4回）

特集『組み込みシステムの世界へようこそ！』についてのアンケートの結果

Q1　あなたは組み込み技術者ですか？
①はい（50%）
②いいえ（50%）

Q2　組み込み関係の記事で読みたいものは何ですか？
①ハードウェア設計全般（22%）
②特有のチップ解説（7%）
③組み込みOSの使い方（26%）
④プロトコル・スタックの解説（15%）
⑤アプリケーション開発（11%）
⑥システム設計（19%）
⑦その他（具体的に）（0%）

Q3　組み込み技術は今後どうなると思いますか？
①このまま専用チップ・専用OSが使われ続ける（22%）
②PCとの差異がなくなる（13%）
③ハイエンドの技術が降りてくる（35%）
④日本が組み込みの最先端であり続ける（17%）
⑤海外への技術移転が進む（13%）
⑥その他（0%）

特集担当デスクから

☆組み込みLinuxなどの解説記事で，そのプラットホームがMIPSプロセッサだったことは何度かありましたが，特集すべてを使ってMIPSプロセッサの解説を行ったのは，今回が初めてになるでしょう．
☆MIPSといえばRISCプロセッサの代名詞でもあり，シリコングラフィックス社のワークステーションなどEWSに搭載され進化してきましたが，EWS分野が縮小してからは，組み込み分野でますます勢力を伸ばしているといったところです．
☆コアをライセンスして各半導体メーカがデバイスを製造するという構図は，ARMに近いといえます．ARMとMIPSの大きな違いは，ライセンシがコアの設計に手を入れられるか入れられないかにあります．それゆえ，厳密に互換性が確保されているARMと，各社まちまちのMIPSと見られがちです．
☆とはいえ，ARMでも周辺コントローラは各社各様のものを内蔵しているとなれば，トータルで互換性を考えると現実的にARMもMIPSもそれほど変わりない…といったところでしょうか．逆に，プロセッサ＋チップセットという構成であれば，東芝のプロセッサにNECのチップセットという組み合わせで動いているT-Engineもあるわけで，MIPSの手法も非常におもしろいのではないかと感じます．
☆今回の特集の企画のため打ち合わせしていると「え？ あれもMIPSなんですか！」というような話が次々と…（ナイショにしている物もあるのでここでは書けませんが（汗））．有名どころの製品にも人知れず搭載されている事実を改めて思い知らされました．

Interface

次号予告

2004年8月号は
6月25日発売です

次世代TRON——
T-Engine & T-Kernel

TRONアーキテクチャは，従来からμITRONとして組み込みの世界で普及してきた．μITRONは小規模な組み込みシステムにおいてコンパクトな仕様と実装のしやすさなどが評価されてきた．

しかし近年の組み込み機器の多機能化により，組み込み向けOSに対する要望が高まり，メモリ保護機能や動的なメモリ管理など，高度な機能が要求されるようになった．また，ITRONの「弱い標準化」の考えかたにより，デバイス・ドライバなどのミドルウェアが統一できていないという問題点も指摘されている．

そこでハードウェアをある程度統一化し，そのうえにITRONベースの強固なOSを載せるという「T-Engine」と「T-Kernel」の仕様が策定された．T-Engineは，そのまま製品に組み込んで使えるだけでなく，開発プラットホームとしても使えるなど，幅広い応用が可能な製品である．そこで次回の特集では，T-EngineとT-Kernelについて徹底解説を行う．

★次号には，記事関連ファイルなどが満載されたCD-ROM「InterGiga No.33」が付属します！

編集後記

●政治都市・北京とビジネス都市・上海を結ぶ新幹線がTGV方式に落ち着きそうだ．北京-上海間は約1300km．その間の駅は天津・南京など27駅．時速300km/hで飛ばして6時間．将来は350km/hで4時間30分．東京から博多まで1175km，のぞみで約5時間．この距離はやっぱり飛行機でしょう．メンテも大事．（檀）

●D○C○M○なんかは次々と新しい端末が出るのに，私が使ってるキャリアの端末の選択肢の寂しさといったらもう…（涙）もっとも機種変更するつもりもないんでどうでも良かったんですが，ついに先日1年ぶりの新機種発表でその仕様にちょっと心が…いや，これの対抗機種がM下あたりから出るのを待て！？（M）

●巣鴨は不思議なところで，狭い割には武道の道場がたくさんある．空手が三つ，ブラジリアン柔術が一つ，キックボクシングが一つ．それらの中には名の通った先生が教えている道場がいくつもある．また，15分も歩けば，元キック・ボクシング・チャンピオンが経営している食堂が！格闘好きには居心地が良い．（＝IO）

●会社帰りに秋葉原．ガード下のパーツ屋から，ジャンク屋，果てはメイド喫茶や精神世界系グッズ店まで，いろいろな意味で怪しい店が軒を連ねる街ですが，ようやく駅前の空き地にもビルが建ち始めた模様．普通の店でなく，やはり秋葉原でなければ買えないような「何か」を扱うテナントが入るといいですね．（み）

●うあぁぁ～～！困った！！このところ，パソコンの調子がものすごく悪い．1日に40回くらいはフリーズするし，起動できないアプリケーションもある（←しかも，エディタだよ）．OSを再インストールしようとしたところ，その真っ最中にもフリーズした．どうすればいいんだあぁっ！！！いよいよ修理かっ？！！！（もみ）

●もともと記念日・誕生日は忘れがちで親しい人の誕生日すら忘れたりするのですが，やはり今年も（!?）だいじな記念日を忘れてました．思い出したのも相手に言われてから…．正直，聞かれたときには何のことかわかりませんでした．こんな私ですが，皆さんも広い心で接してください．（T_T）（Y2）

●TVで紹介された店で餃子を食べた．しかし，それほどおいしいかは疑問．別の日，人気店といわれる店にて．土曜の夜なのに…ガラガラ．水餃子，焼き餃子，揚げ餃子の各種を注文．水餃子と焼餃子はおいしかったが，広い店内で，不思議なほど客がいなくて落ち着かず逃げるように店を出た．（太陽熱）

●ペットボトル飲料水を買ったら，サッカー選手のフィギュアが付いていました．興味がないので捨てようとしましたが，オマケとはいえ実在する人間の姿形をした人形を簡単に捨ててよいものなのかと，一瞬迷いが．結局「ゴメンね」と思いつつ捨てましたが，オマケフィギュアの祟りなんてありませんよね…．（と）

お知らせ

■読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■コピー・サービスのご案内

本誌バックナンバの掲載記事については，在庫（原則として24か月分）のないものに限りコピー・サービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

●コピー料金（税込み）
1ページにつき100円

●発送手数料（判型に関わらず）
1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円

●送付金額の算出方法
総ページ数×100円＋発送手数料

●入金方法
現金書留か郵便小為替による郵送

●明記事項
雑誌名，年月号，記事タイトル，開始ページ，総ページ数

●宛て先
〒170-8461　東京都豊島区巣鴨1-14-2
CQ出版株式会社　コピー・サービス係
（TEL：03-5395-4211，FAX：03-5395-1642）

■お問い合わせ先のご案内

●在庫，バックナンバ，年間購読送付先変更に関して
販売部：03-5395-2141

●広告に関して
広告部：03-5395-2133

●雑誌本文に関して
編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版（株）　2004　振替　00100-7-10665
2004年7月号　第30巻　第7号　通巻第325号）
2004年7月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏　村上真紀　山口光樹　大竹友美　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森　祐子
広告／澤辺　彰　中元正夫　菅原利江

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122　FAX／（03）5395-2127
広告部（03）5395-2133　URL http://www.cqpub.co.jp/interface/
販売部（03）5395-2141　E-mail supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社


日本ABC協会加盟誌
（新聞雑誌部数公査機構）　ISSN0387-9569